НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____20__ р.

Дипломний проект

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

спеціальності 123 «Комп'ютерна інженерія»

на тему: «Структура, метод побудови незворотних булевих перетворень для криптографічно строгої ідентифікації віддалених користувачів та програмні засоби його реалізації»

Виконала:
студентка IV курсу, групи ІО-62
Анна КЛИМОВА _____

Керівник:
Доцент, к.т.н.,
Олександр МАРКОВСЬКИЙ _____

Консультант з нормоконтролю:
Професор, д.т.н.,
Валерій Сімоненко _____

Рецензент:
Декан ФПМ, проф., д.т.н.
Іван ДИЧКА _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2020 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМ. ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність - 123 «Комп’ютерна інженерія»

Освітньо-професійна програма «Комп’ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО

“___”_____2020р.

ЗАВДАННЯ
на дипломний проект студента
Климової Анни Сергіївни

1. Тема проекту «Структура, метод побудови незворотних булевих перетворень для криптографічно строгої ідентифікації віддалених користувачів та програмні засоби його реалізації»_____

керівник проекту _____ к.т.н доцент Марковський О.П._____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "07" травня 2020 року №1081-с

2. Термін подання студентом проекту
_____

3. Вихідні дані до проекту __технічна документація,

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розробка структури і методу отримання булевих перетворень, що володіють властивостями незворотності та неоднозначності. Створення програмної системи для практичної реалізації способу ідентифікації абонентів, заснованого на концепції «нульового знання» та розробленому методі. Дослідження отриманого алгоритму.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)

Принципова схема, функціональна схема та структурна схема

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
нормоконтроль	д.т.н., проф. Сімоненко В.П.		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту	Строк виконання етапів проекту	Примітка
1	*Затвердження теми роботи*	*1.09.2019*	
2	*Вивчення та аналіз завдання*	*2.09.2019-01.03.2020*	
3	*Розробка архітектури та загальної структури програми*	*02.03.2020-01.04.2020*	
4	*Програмна реалізація*	*02.04.2020-20.04.2020*	
5	*Дослідження програмного продукту*	*21.04.2020-01.05.2020*	
6	*Оформлення пояснювальної записки*	*02.05.2020-25.05.2020*	
7	*Передзахист*	*26.05.2020*	
8	*Захист*		

Студент _____ Анна КЛИМОВА

Керівник проекту _____ Олександр МАРКОВСЬКИЙ

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проект	2	
2	А4	ІАЛЦ 467200.002 ТЗ	Технічне завдання	3	
3	А4	ІАЛЦ 467200.003 ПЗ	Пояснювальна записка	69	
4	А1	ІАЛЦ 467200.004 ТК	Структурна схема	1	
5	А1	ІАЛЦ 467200.004 ТК	Функціональна схема	1	
6	А1	ІАЛЦ 467200.004 ТК	Принципова схема алгоритму	1	

Зм.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.4672000.001 ВП		
Розробив		Климова А.С.			Структура, метод побудови незворотних булевих перетворень для криптографічно строгої ідентифікації віддалених користувачів та програмні засоби його реалізації Відомість дипломного проекту	Лит.	Аркуш 1 / Аркушів 1
Перевірив		Марковський.О.П.					
Н.Контр.		Сімоненко В.П.				НТУУ "КПІ" ім.. Ігоря Сікорського ФІОТ, гр. ІО-62	
Затвердив		Дичка І.А.					

ТЕХНІЧНЕ ЗАВДАННЯ

Київ – 2020 року

ЗМІСТ

					ІАЛЦ.467200.002 ТЗ		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Климова А.С.			Структура, метод побудови незворотних булевих перетворень для криптографічно строгої ідентифікації віддалених користувачів та програмні засоби його реалізації Технічне завдання	Літ.	Арк. 1 / Аркушів 3
Перевір.		Марковський О.П.					
Н. контр.		Сімоненко В.Г.				НТУУ «КПІ»ім. Ігоря Сікорського, ФІОТ ІО-62	
Затверд.		Дичка І.А.					

1. Найменування та сфера застосування

Технічне завдання розроблено для написання дипломної роботи на тему «Структура, метод побудови незворотних булевих перетворень для криптографічно строгої ідентифікації віддалених користувачів та програмні засоби його реалізації». Сфера застосування являє собою багатокористувальницькі системи, в яких необхідно ідентифікувати користувачів.

2. Обґрунтування розробки

Основою для розробки є завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», що затверджено кафедрою обчислювальної техніки Національного Технічного Університету України «Київського Політехнічного Інституту ім. Ігоря Сікорського».

3. Мета та призначення розробки

Метою роботи є вивчення існуючих методів ідентифікації віддалених користувачів, розробка методу отримання мулевих перетворень, що буде мати властивість неоднозначності, і створення системи, що використовуватиме розроблений спосіб ідентифікації абонентів, заснований на концепції «нульових знань».

4. Джерела розробки

Вхідними даними для роботи є теоретичні матеріали по темі «Схеми ідентифікації користувачів», публікації в періодичних виданнях, довідники, статті в Інтернеті за необхідною тематикою.

5. Технічні вимоги

Основні вимоги до устаткування:

– об’єм оперативної пам’яті — 256 Мб, рекомендований об’єм — 1024 Мб та більше;
– мінімальний об’єм вільного дискового простору — 15 мегабайт, рекомендований об’єм — 100 мегабайт та більше;
– монітор із підтримкою мінімального дозволу екрану 800 на 600 пікселів; для зручної роботи рекомендується дозвіл 1280 на 1224 пікселів та більше.

Вимоги до програмного забезпечення:

– операційна система Microsoft Windows..

Анотація

Метою бакалаврської дипломної роботи є розробка структури і методу отримання булевих перетворень, що володіють властивістю неоднозначності, створення програмної системи для практичної реалізації способу ідентифікації абонентів, заснованого на концепції «нульового знання» та розробленому методі. Дослідження отриманого алгоритму.

Ключові слова: ідентифікація віддалених абонентів, концепція «нульових знань», захист інформації, булеві функціональні перетворення.

Аннотация

Целью бакалаврской дипломной работы является разработка структуры и метода получения булевых преобразований, обладающих свойством неоднозначности, создание программной системы для реализации способа идентификации абонентов, основанного на концепции «нулевого знания» и разработанном методе. Исследование полученного алгоритма.

Ключевые слова: идентификация удаленных абонентов, концепция «нулевых знаний», защита информации, булевы функциональные преобразования.

Annotation

The purpose of work for a Bachelor's Degree is to develop a structure and method for obtaining Boolean transformations that have the property of ambiguity, to create a software system for implementing the method of identifying subscribers based on the concept of “zero knowledge” and the developed method. Study of the resulting algorithm.

Key words: identification of remote subscribers, the concept of “zero knowledge”, information protection, Boolean functional transformations.

Пояснювальна записка
до дипломного проекту
на тему: «Структура, метод побудови незворотних булевих перетворень для криптографічно строгої ідентифікації віддалених користувачів та програмні засоби його реалізації»

Київ – 2020 року

ЗМІСТ

ПЕРЕЛІК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

ОС - Операційна система

CLR - (англ. Common Language Runtime) середовище виконання часу, що запускає код і надає послуги, що полегшують процес розробки, яке пропонує .NET Framework.

WPF - (англ. Windows Presentation Foundation) частина системи платформи .NET і являє собою підсистему для побудови графічних інтерфейсів.

MVC (англ. model-view-controller) - спосіб організації коду, який передбачає виділення блоків, що відповідають за вирішення різних завдань. Один блок відповідає за дані додатки, інший відповідає за зовнішній вигляд, а третій контролює роботу додатка.

MVVM (англ. model-view-view-model) - архітектурний паттерн проектування, орієнтований на сучасні платформи розробки, такі як Windows Presentation Foundation, Silverlight від компанії Microsoft, ZK framework.

API (англ. Application programming interface) - набір визначень взаємодії різнотипного програмного забезпечення.

XAML (англ. eXtensible Application Markup Language) - мова розмітки, яка використовується для ініціалізації об'єктів в технологіях на платформі .NET. Стосовно до WPF (а також до Silverlight) дана мова використовується перш за все для створення призначеного для користувача інтерфейсу декларативним шляхом.

EF (англ. Entity Framework) - це об'єктно-реляційний маппер (ORM), який дозволяє розробникам .NET працювати з базою даних за допомогою об'єктів .NET. Це виключає необхідність більшості коду доступу до даних, який розробникам зазвичай потрібно писати.

ВСТУП

У сучасному світі питання надійності ідентифікації віддалених абонентів є одним з найважливіших для забезпечення результативного контролю за доступом до персональних даних користувачів.

Необхідність підвищення ефективності ідентифікації віддалених абонентів на сьогодні обумовлена швидким розвитком інтегрованих систем зберігання й обробки даних, значним зростанням випадків несанкціонованого доступу до персональних даних користувачів. Багато видів комерційної діяльності, фінансові операції, обмін документами здійснюється через відкриті і вразливі засоби зв'язку, наприклад, через Інтернет. Заміна традиційного способу ведення ділових операцій електронним, з використанням мережі Інтернет, пов'язана зі значними ризиками для організацій. Надзвичайно важливо, щоб партнери довіряли один одному, а повідомлення не фальсифікувалися. Для цього необхідно забезпечити аутентифікацію за допомогою прикладних криптографічних засобів.

Криптографія нині проникла усюди, від веб-браузерів до почтових програм, на мобільні телефони, банківські карти, наші автомобілі та навіть медичні імплантати. Для того щоб забезпечити як великим, так і малим підприємствам повний захист їх даних при використанні web-ресурсів, їм необхідно гарантувати, що корпоративні споживачі і стратегічні партнери (постачальники, контрагенти і консультанти) надійно ідентифіковані, їх доступ до мереж правомірний і безпечний, а використовувані інформаційні канали віддаленого доступу захищені належним чином. Контроль доступу користувачів до ресурсів корпоративної мережі повинен здійснюватись відповідно до політики безпеки організації, до якої належить дана мережа. Ефективне розмежування доступу до мережевих ресурсів може бути забезпечено тільки при надійній аутентифікації користувачів. Вимоги до надійності аутентифікації віддалених користувачів повинні бути особливо високими. Це обумовлено тим, що при взаємодії з фізично віддаленими

користувачами значно складніше забезпечити доступ до мережевих ресурсів тільки для тих користувачів, які мають на це повноваження. На відміну від локальних користувачів, віддалені користувачі не проходять процедуру фізичного контролю при допуску на територію організації.

Проблема захисту даних завжди була актуальною, адже будь-який алгоритм можна зламати. Питання лише у вартості і часі та інтересі зловмисника, а отже персональні дані користувачів знаходяться в безпеці лише якийсь проміжок часу. Саме тому ця галузь потребує все нових і нових рішень.

Про метод організації системи індентифікації віддалених користувачів і піде мова в даному дипломному проекті.

РОЗДІЛ 1.

АНАЛІЗ СУЧАСНИХ СИСТЕМ ІДЕНТИФІКАЦІЇ ВІДДАЛЕНИХ АБОНЕНТІВ

1.1 Аналіз вимог до засобів ідентифікації віддалених користувачів

Коли ми чуємо слово криптографія[1,2,3], нашими першими асоціаціями можуть бути шифрування, безпечний доступ до веб-сайтів, смарт-карт для банківських додатків.

Криптографія на сьогоднішній день тісно пов'язана із сучасним електронним зв’язком. Однак, вона бере свій початок дуже давно, ще приблизно з 2000 року до н.е., коли в Стародавньому Єгипті використовувались нестандартні «таємні» ієрогліфи. З тих часів криптографія використовувалася в тій чи іншій формі в багатьох, якщо не в більшості, культур, що розвивали писемну мову.

Надалі слід проналізувати сферу криптографії. Якщо заглибитися в дану сферу, досить очевидно, що більш загальним терміном є криптологія., а не криптографія. Власне криптологія вже розбивається на дві галузі: криптографію та криптоаналіз. Перше – це ніщо інше, як наука, що займається вивченням таємного письма з метою приховування сенсу повідомлення. Друге – наука, яка спрямована на злом криптосистем. Під криптосистемою ми розуміємо певну комплексну модель, що використовує двостороння крипто перетворення над даними будь-якого об’єму, алгоритм здійснення . Криптоаналіз вивчає математичні методи обходу таких систем з метою отримання засекреченої інформації. З першого погляду можна подумати, що таке порушення цілісності даних призначене для розвідувальної спільноти чи, можливо, навіть організованого злочину, і не слід включати його до серйозної класифікації наукових дисциплін. Однак, більшість криптоаналізів наразі проводиться поважними дослідниками в академіях , а сама наука спрямована на забезпечення безпеки. Без людей, які

намагаються зламати запропоновані крипто-методи, ніколи не можна переконатися в їх безпеці. Оскільки криптоаналіз – єдиний спосіб переконатися в надійності криптосистеми, він є частиною криптології.

Власне криптографія розбивається ще на три основні галузі:

- симетричні алгоритми: дві сторони обмінюються повідомленнями. Вони мають метод шифрування та дешифрування, для цього використовують деякий секретний ключ. Вся криптографія з давніх часів і до 1976 року ґрунтувалася виключно на симетричних методах. Симетричні шифри все ще мають широку сферу застосування, особливо для шифрування даних і перевірки цілісності повідомлень.

- асиметричні (або з відкритим ключем) алгоритми. У 1976 році Уітфілд Діффі, Мартін Хеллман і Ральф Меркле представили новий тип шифрів. У криптографії з відкритим ключем користувач має секретнй ключ, як в симетричній криптографії, але також і відкритй ключ. Асиметричні алгоритми можуть використовуватися для таких додатків, як цифрові підписи і встановлення ключів, а також для класичного шифрування даних.

- строго кажучи, хеш-функції, які саме будуть цікавити нас в ході виконання дипломного проекту, форма третього класу алгоритмів, але в той же час вони поділяють деякі властивості із симетричними шифрами.

Криптографічні протоколи мають справу з додатком криптографічних алгоритмів. Симетричні та асиметричні алгоритми можна розглядати як будівельні блоки, за допомогою яких такі додатки як захищений Інтернет-зв'язок може бути здійснений. Схема безпеки транспортного шару (TLS), який використовується у кожному веб-переглядачі, є прикладом криптографічного протоколу.

У більшості криптографічних застосувань у практичних системах симетричні та асиметричні алгоритми (а часто також і хеш-функції) використовуються разом. Це іноді називають гібридними схемами.

Причиною використання обох групп алгоритмів є те, що кожен має конкретні сильні та слабкі сторони.

Однією з основних задач криптографії є двостороння інтерактивна гра, в якій один з учасників доводить іншому істинність твердження, не розкриваючи суті доказу. В такому випадку перевіряюча сторона не може самостійно оцінити істинність твердження, оскільки йому не відома інформація, що доступна тому, хто доводить. Така гра називається протоколом інтерактивного доказу або IP-протоколом (англ. interactive proof, не плутати з протоколом IP мережі Інтернет). Доказ, що здійснюється IP-протоколом, можна назвати "секретним доказом". Секретність цього доказу полягає в тому, що, по-перше, перевіряюча сторона, переконавшись в істинності доказуваного твердження, не здатна самостійно повторити доказ, і, по-друге, після завершення протоколу ніхто зі сторонніх не здатний зрозуміти жодного повідомлення, якими обмінювалися обидві сторони.

Уявімо собі, що твердження, яке необхідно довести не розкриваючи суті доказу, є вирішенням якої-небудь відомої невирішеної математичної задачі В цьому випадку сторона, що доводить, яка побоюється плагіату, може побажати приховати технічні деталі доказу від потенційно нечесного рецензента. Для цього вона повинна провести "секретний" доказ, переконавши рецензента (він тут грає роль перевіряючою сторонни в IP-протоколі) в коректності висновків, не даючи ніякої додаткової інформації.

У багатьох реальних додатках існують набагато серйозніші підстави для проведення "секретних" доказів. Як правило, IP-протоколи застосовуються для аутентифікації сутностей. На відміну від звичайних протоколів аутентифікації, в яких користувачі ставлять цифрові підписи, в IP-протоколі сторона, яка доводить, яка підлягає аутентифікації, не бажає, щоб повідомлення стали доступними будь-кому, крім перевіряючої сторони, і виконує "секретну" аутентифікацію. Крім того, IP -протоколи часто застосовуються для того, щоб довести, що частина прихованої інформації має

певну структуру. Це необхідно в деяких секретних додатках (наприклад, при проведенні електронних аукціонів), в яких прихований номер (лот) повинен знаходитися в допустимих межах.

Розглядаючи IP -протоколи, необхідно визначитись з таким питанням "Скільки інформації отримає перевіряюча сторона в ході інтерактивного доказу? "Ідеальною відповіддю на це запитання було б "нуль". IP -протокол, що володіє такою властивістю, називається протоколом з нульовим розголошенням або ZK-протокол(англ. zero-knowledge).

1.2 Ідентифікація абонентів на основі концепції “Нульових знань” з використанням булевих перетворень спеціальних класів

Найбільш перспективним напрямком підвищення ефективності ідентифікації абонентів багатокористувацьких систем є розширення практичного використання технологій, які, з теоретичної точки зору, забезпечують найбільш високий рівень захисту від несанкціонованого доступу до ресурсів систем. Однією з найбільш прогресивних таких технологій є криптографічна концепція “нульових знань”[2].

Однак, на практиці, суттєвим фактором, що обмежує використання теоретичної концепції нульового знання є те, що для її реалізації необхідні значні об'єми обчислювальних ресурсів. Це пов'язано з тим, що в існуючих її реалізаціях використовуються вельми ресурсомісткі мультиплікативні модулярні операції, що виконуються над числами, кількість розрядів яких в сотні разів перевищують розрядність процесора. В умовах динамічного збільшення кількості абонентів багатокористувацьких систем, що обганяє зростання продуктивності комп'ютерних систем, фактор швидкості ідентифікації стає все більш значущим. Тому, для розширення використання прогресивних технологій ідентифікації, що забезпечує найбільший рівень захисту від незаконного доступу, необхідно знайти можливості підвищення продуктивності їх обчислювальної реалізації.

Для радикального підвищення швидкості ідентифікації абонентів на основі прогресивної концепції "нульових знань" необхідно використовувати при її реалізації незворотні перетворення, обчислювальна реалізація яких вимагає значно менших обчислювальних ресурсів в порівнянні з перетвореннями, що базуються на аналітично нерозв'язних задачах теорії чисел. Одними з таких перетворень є булеві нелінійні перетворення, які, як відомо, є аналітично незворотними і обчислювальна реалізація яких має відносно невелику обчислювальну складність.

Для того, щоб підвищити швидкість ідентифікації за рахунок переходу від незворотних перетворень на основі теорії чисел до булевих незворотних перетворень, необхідно розробити спосіб використання останніх в рамках концепції "нульового знання", виявити властивості таких перетворень, що забезпечують можливість їх застосування, а також розробити метод отримання булевих перетворень, які володіють зазначеними властивостями.

1.3 Огляд сучасних ситем ідентифікації віддалених користувачів та аналіз їх вразливості

Коли мова йде про ідентифікацію віддалених абонентів, найчастіше з цим асоціюються хеш-функції. У зв’язку зі своїми властивостями односторонньості та унікальності вони ідеально підходять для цього завдання, тож надалі ми розглянемо їх детальніше.

Функція хешування - це детермінована функція, яка відображає рядок бітів довільної довжини в хешоване значення, що представляє собою рядок бітів фіксованої довжини. Позначимо через h хеш-функцію, що повертає рядок фіксованої довжини | h |. Бажано, щоб функція h володіла такими властивостями:

1. Перетворення перемішування. При будь-якому аргументі x хешоване значення h(x) має не відрізнятись з обчислювальної точки зору від рядка бітів, рівномірно розподілених в інтервалі [0, 2).

2. Запобігання колізій. Пошук двох величин x і y, що задовольняють умовами $x \neq y$ і $h(x) = h(y)$, повинен являти собою нерозв'язну задачу. Для того, щоб це припущення було розумним, необхідно, щоб область значень h була великою. Число | h | не повинно бути менше 128 і, як правило, так само 160.

3. Складність обчислення прообразів. Завдання обчислення вхідного рядка x по заданому хешованому значенню $h = h(x)$ повинна бути нерозв'язним обчислювальним завданням. У цьому випадку також необхідно, щоб область значень p була досить великою.

4. Практична ефективність. Обчислення значення h(x) має бути обмежене поліномом невеликого ступеня (в ідеальному випадку - лінійним), що залежить від розміру рядка x.

Властивості перемішування і запобігання колізій можна забезпечити за допомогою тих самих функцій, які використовувалися в алгоритмах блочного шифрування. Складність обчислення прообразів досягається за рахунок стиснення даних, при якій втрачається призводить до втрати інформації, і, внаслідок цього, утруднюється обчислення оберненої функції.

Хеш-функції можуть широко використовуються в системах для побудови кодів аутентифікації, а тому є гарним прикладом готових рішень для системи, що розробляється в рамках дипломного проекту. Розглянемо кілька прикладів:

● **RC-5**

Алгоритм розроблений відомим в криптології Рональдом Рівестом - одним з розробників асиметричною системи RSA[2] і одним із засновників однойменної фірми (RSA Data Security), яка, без сумніву, є одним зі світових лідерів ринку засобів криптографічного захисту інформації. Абревіатура RC позначає, за різними джерелами, або Rivest Cipher, або Ron's Code, тобто, в сукупності, «шифр Рона Рівеста» .

Аналогічно попереднім алгоритмам шифрування Рона Рівеста RC2 і RC4, алгоритм RC5[4] отримав досить широке поширення; за кількістю користувачів в світі він стоїть в одному ряду з такими відомими алгоритмами, як IDEA і Blowfish.

На перетвореннях, які використовуються в RC5, заснована подальша розробка компанії RSA - алгоритм RC6 , який став фіналістом конкурсу AES за вибором нового стандарту шифрування США; алгоритм RC6 не переміг в конкурсі, але, мабуть, перевершить свого предка по широті використання.

Схема одного раунда алгоритму представлена на рис. 1.2.

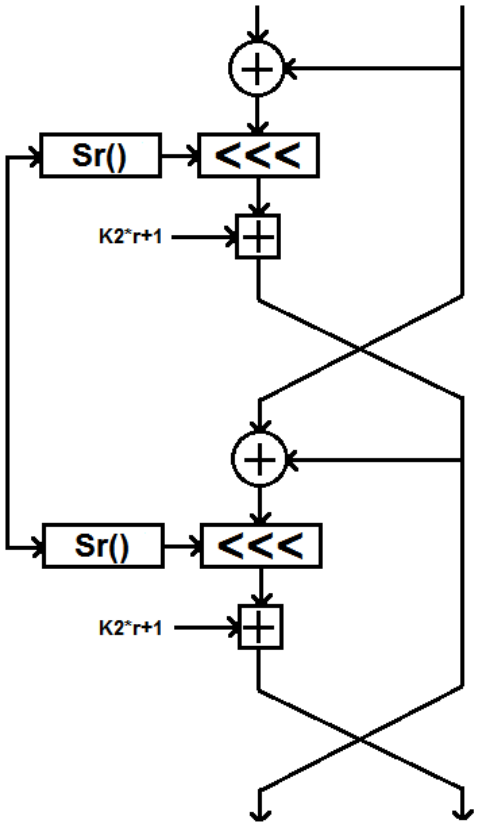


Рис. 1.2. Один раунд алгоритму RC-5

Структура алгоритму:

Аналогічно, наприклад, алгоритму SHARK, частина основних параметрів алгоритму RC5 є змінними. Як пише автор алгоритму «RC5 - це кілька різних алгоритмів», оскільки, крім секретного ключа, параметрами алгоритму є наступні :

- розмір слова w (в бітах); RC5 шифрує блоками по два слова; допустимими значеннями w є 16, 32 або 64, причому 32 є рекомендованим;

- кількість раундів алгоритму R - як значення допустимо будь-яке ціле число від 0 до 255 включно;

- розмір секретного ключа в байтах видання - будь-яке ціле значення від 0 до 255 включно.

Найбільш часто для уточнення параметрів алгоритму, використовуваних в його конкретної реалізації, застосовується позначення RC5-w / R / b ', наприклад, RC5-32 / 12/16 позначає алгоритм RC5 з 64-бітовим блоком, 12 раундами і 128-бітовим (16 -байтним) ключем. Таку комбінацію параметрів Рівест рекомендує в якості основного варіанту алгоритму.

На думку автора алгоритму, змінні параметри розширюють сферу використання алгоритму, а також дозволяють сильно скоротити витрати при необхідності переходу на більш сильний варіант алгоритму - на відміну від DES (основна проблема якого - короткий 56-бітний ключ), в програмної або апаратної реалізації RC5, підтримуючої змінні параметри, легко було б замінити ключ довшим, таким чином усунувши проблему. Ось що пише про це Рон Рівест: «Фіксовані параметри можуть бути не менш небезпечні, оскільки вони не можуть бути поліпшені при необхідності.

Автор передбачив і проблему сумісності реалізацій RC5 з різними параметрами - кожне зашифроване повідомлення рекомендується випереджати заголовком, що містить список значень основних параметрів алгоритму - передбачається, що в цьому випадку для розшифрування повідомлення слід встановити параметри з заголовка, після чого (при наявності коректного ключа) повідомлення легко буде розшифровано.

● SHA-256

SHA-256[5] (рис 1.3) являє собою односпрямовану функцію для створення цифрових відбитків фіксованої довжини (256 біт, 32 байт) з вхідних даних розміром до 2,31 ексабайт (2^{64} біт) і є окремим випадком

алгоритму з сімейства криптографічних алгоритмів SHA-2 (Secure Hash Algorithm Version 2) опублікованим АНБ США в 2002 році.

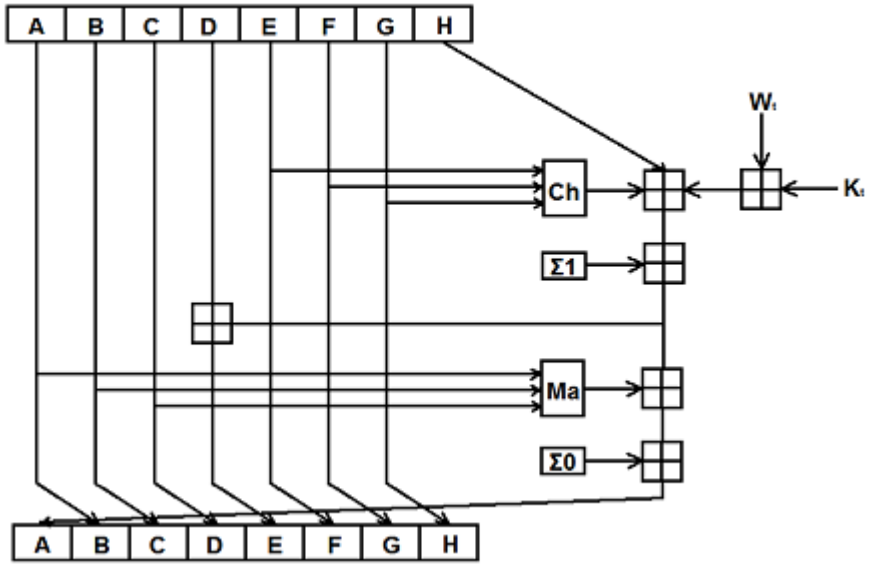


Рис. 1.3. Алгоритм SHA-256

Хеш-функції сімейства SHA-2 побудовані на основі структури Меркле - Дамгарда.

Оригінал тексту після доповнення розбивається на блоки, кожен блок - на 16 слів. Алгоритм пропускає кожен блок повідомлення через цикл з 64 ітераціями. На кожній ітерації 2 слова перетворюються, функцію перетворення задають інші слова. Результати обробки кожного блоку складаються, сума є значенням хеш-функції. Так як ініціалізація внутрішнього стану проводиться результатом обробки попереднього блоку, то немає можливості обробляти блоки паралельно. Графічне представлення однієї ітерації обробки блоку даних:

На поточний момент відомі методи для конструювання колізій до 31 ітерації. З огляду на алгоритмічної схожості SHA-2 з SHA-1 і наявності в останньої потенційних вразливостей прийнято рішення, що SHA-3 буде базуватися на зовсім іншому алгоритмі. 2 жовтня 2012 року NIST затвердив в якості SHA-3 алгоритм Keccak.

● **RIPEMD-160**

RIPEMD-160[6] є 160-бітної криптографічного хеш-функцією, розробленої Хансом Доббертіном (Hans Dobbertin), Антоном Босселарсом (Antoon Bosselaers), і Бартом Пренелом (Bart Preneel). Він призначений для заміни менш безпечних 128-бітних хеш-функцій MD4, MD5 і RIPEMD. MD4 і MD5 були розроблені Рональдом Ривестом (Ronald Rivest) для RSA Data Security, в той час як RIPEMD був розроблений для проекту RIPE (RACE Integrity Primitives Evaluation, 1988-1992) Європейського співтовариства.

Три вагомі причини, чому необхідна така заміна:

1. 128-бітний хеш більше не забезпечує достатнього захисту. Пошук колізій методом перебору (брутфорс) для 128-бітного хеша вимагає 264 або приблизно 2.1019 обчислень. У 1994 році Пол ван Оршот (Paul van Oorschot) і Майк Вінер (Mike Wiener) показали, що такий брутфорс можна зробити менше ніж за місяць з $ 10 мільйонами доларів інвестицій. Ця вартість брутфорса, як очікується, зменшується вдвічі кожні 18 місяців.

2. У першій половини 1995 року Ханс Доббертін знайшов колізії для версії RIPEMD, обмежуючись двома раундами з трьох. Використовуючи схожу методику, Ханс восени 1995 року знайшов колізії (всі 3 раунди) для MD4. Атака на MD4 займає приблизно кілька секунд на комп'ютері, і до сих пір залишає деяку свободу у виборі повідомлення, виключаючи MD4 як стійку до колізій хеш-функцію. Незабаром після цього, навесні 1996 року, Ханс також виявив колізії для функції стиснення MD5. Ця атака викликає серйозні сумніви в тому, що MD5 є крипостійким. RSA Data Security, для якої Рональд Ривест розробив MD4 і MD5, рекомендує: MD4 не повинна більше ніде використовуватися, а MD5 не повинна використовуватися в майбутніх додатках, для яких потрібна хеш-функція, стійка до колізій.

3. В кінці сесії Crypto 2004 було оголошено, що Xiaoyun Wang, Dengguo Feng, Xuejia Lai і Hongbo Yu знайшли колізії для MD4, MD5, RIPEMD, і 128-бітної версії HAVAL.

RIPEMD-160 являє собою поліпшений варіант RIPEMD з результуючим 160-бітовим хешем, і передбачається, він буде безпечним протягом наступних десяти років або більше. Конструктивні особливості засновані на накопиченому досвіді попередніх хеш-функцій MD4, MD5 і RIPEMD. Як і його попередники, RIPEMD-160 налаштований для 32-розрядних поколінь процесорів, які, на думку авторів, будуть залишатися поширеними в найближче десятиліття.

RIPEMD-128 являє собою лише заміну оригінальної RIPEMD (або MD4 і MD5, якщо на те пішло), що обчислює 128-розрядні хеш-значення. Варто нагадати, що оригінальний алгоритм RIPEMD теж має 128-бітний хеш і в ньому були знайдені вразливості. Так як 128-бітний хеш не дає достатнього захисту протягом наступних десяти років, то в додатках, що використовують 128-бітові хеші, слід задуматися про перехід на 160-бітну хеш-функцію.

RIPEMD-256 і RIPEMD-320 є додатковими розширеннями, відповідно, для RIPEMD-128 і RIPEMD-160, і відрізняються подвоєною довжиною дайджесту, що зменшує ймовірність колізій, але при цьому функції не є більш крипостійкою.

Використання RIPEMD-160 не обмежена будь-якими патентами.

Значними недоліками хуш-функцій є:

- наявність колізій. Колізія – співпадання хеш-сум різних паролей. Це відбувається через те, що вхідне повідомлення може бути якої завгодно довжини, а хеш-сумми – однакової.

- райдужні таблиці. Це таблиці вже порахованих хеш-сум. Якщо пароль користувача є досить простим, його можна знайти по хешу в такій таблиці.

Наявні також методи ідентифікації віддалених абонентів, що базуються на концепції "нульових знань". Найбільш відомими такими методами є:

- FFSIS(Feige Fiat Shamir Identification Sheme); -

- Шнора(Schnorr);

- Гіллоу-Квіскватера (Guillou-Quisquater).

Базові обчислювальні операції для FFSIS є $A^2B \ mod \ m$, а для методів Шнора і Гіллоу-Квіскватера - $A^eB^v \ mod \ m$. Враховуючи те, що для забезпечення потрібного рівня захищеності розрядність чисел становить 2048 або 4096, швидкість ідентифікації обмежується значною обчислювальною складністю таких операцій.

ВИСНОВКИ ДО РОЗДІЛУ 1

В даному розділі було проведено аналіз вимог до засобів ідентифікації віддалених користувачів, короткий огляд концепції “нульових знань”, а також було розглянуто сучасні алгоритми, що можуть бути використані для ідентифікації віддалених абонентів.

В результаті проведеного дослідження було виявлено недоліки та вразливості цих алгоритмів:

- наявність колізій;
- вже прораховані хеш-суми;
- велика обчислювальна складність алгоритмів.

Через зазначені вище недоліки, багатокористувальницькі системи досі не мають надійних інструментів для ідентифікації віддалених користувачів. Саме тому необхідно розробити такий алгоритм, що буде позбавлений виявлених вразливостей.

Очевидно, що більш надійними є алгоритми, що спираються на концепцію "нульових знань". Тому було прийнято рішення, що розроблюваний метод буде спиратися саме на неї, але матиме меншну обчислювальну складність у порівнянні з попередніми. Така стратегія дозволить розробити надійний алгоритм, позбавлений недоліків сучасних методів ідентифікації віддалених абонентів та надасть багатокористувальницьким системам надійний інструмент для захисту даних.

РОЗДІЛ 2.

РОЗРОБКА СТРУКТУРИ, МЕТОДУ ІДЕНТИФІКАЦІЇ АБОНЕНТІВ

2.1. Розробка способу ідентифікації абонентів на основі концепції “нульових знань” з використанням незворотних булевих функціональних перетворень

Суть концепції "нульових знань" полягає в тому, що треба перевірити, що хтось має певну інформацію, але при цьому перевіряча сторона не має цієї інформації. Тобто треба перевірити наявність знань непрямим шляхом. Для цього потрібен механізм, який покаже перевіряючому, що той, що перевіряється, знає інформацію. В якості такого механізму пропонується використовувати незворотні булеві функціональні перетворення.

В основі всіх криптографічних алгоритмів захисту інформації лежить аналітично нерозв'язна математична задача. У більшості випадків практичного використання, такі завдання мають вид необоротного перетворення Y = F(X), тобто перетворення, для якого визначена алгоритмічно функція F(X) обчислення в прямому напрямку і не існує аналітичного способу отримання функції Ф (Y) зворотного перетворення X=Ф(Y) за відомою функції F(X). Єдиним способом вирішення таких завдань, тобто знаходження значення X для заданого значення Y при відомій функції F(X) є перебір значень X. Велика частина аналітично нерозв'язних завдань, що лежать в основі сучасних криптографічних алгоритмів відноситься до теорії чисел, і булевої алгебри. Зокрема, до теорії чисел ставляться завдання дискретного логарифмування, на основі яких побудовані більшість алгоритмів несиметричного шифрування (алгоритмів з відкритим ключем), в тому числі, широко використовувані на практиці RSA, El-Gamal, ЕЕС, а також алгоритми цифрового підпису, такі як DSS. В основі досить широкого класу криптографічних алгоритмів лежить аналітично нерозв'язна задача булевої алгебри - відшукання коренів систем нелінійних булевих рівнянь. До цього класу алгоритмів відносяться всі алгоритми блокового

симетричного шифрування, такі як DES, IDEA, Rijndael, а також велика частина хеш-алгоритмів, в тому числі, найбільш поширені на практиці RC-5, SHA і RIPEMD-160.

Основною перевагою алгоритмів побудованих на основі аналітично нерозв'язних задач теорії чисел є існування декількох ключів. Так в алгоритмах шифрування RSA, El-Gamal, EEC, ключі, використовувані для прямого і зворотного перетворень різні. При наявності декількох ключів, частина з них можуть використовуватися як відкриті, а частина - як закриті. Це дозволяє будувати на цій основі істотно більш ефективні механізми захисту інформації та організації доступу до інформаційних ресурсів в порівнянні з алгоритмами, що мають єдиний ключ. Саме тому поява в 1978 р першого алгоритму з "відкритим" ключем - RSA, в основі якого лежить аналітично необоротне перетворення $F(X) = A^X \ mod \ M$, пов'язують з "відкриттям нової ери в технології захисту інформації". У теоретичному плані, існування кількох ключів криптографічного перетворення обумовлено тим, що необоротна завдання є неоднозначною, тобто, існує, принаймні, два ключа X1 і X2, для яких виконується $F(X_1) = F(X_2)$. Основним недоліком алгоритмів захисту інформації, в основі яких лежать аналітично нерозв'язні задачі теорії чисел є низька швидкість їх реалізації, обумовлена високою обчислювальною складністю операцій модулярного експоненціювання над числами, розрядність яких становить тисячі.

Цього недоліку позбавлені алгоритми захисту інформації, в основі яких лежить аналітично нерозв'язна задача булевої алгебри. Рекурсивне обчислення систем булевих функцій може бути організовано досить ефективно як програмними засобами на універсальних процесорах, так і спеціалізованими апаратними обчислювачами. В оціночному плані, швидкість криптографічного обробки даних алгоритмами на основі булевих функцій і на основі модулярной арифметики відрізняється на 3-4 порядки. Однак, алгоритми на основі булевих перетворень мають істотно менші

функціональні можливості, які не дозволяють реалізувати з їх використанням ефективних протоколів захисту інформації, подібно до алгоритмів на основі нерозв'язних задач теорії чисел. Зокрема, використання булевих перетворень не дозволяє реалізувати базові для сучасних технологій захисту інформації концепції несиметричного шифрування даних, цифрового підпису повідомлень, ідентифікації на основі схеми "нульового знання". Одним з факторів, що обумовлюють обмежені функціональні можливості алгоритмів на основі булевих перетворень є однозначність останніх.

Незворотність булевих функціональних перетворень заснована на неможливості аналітичного знаходження коренів нелінійних булевих рівнянь. Єдиним способом вирішення такого завдання є перебір. Як вже зазначалося вище, властивість незворотності нелінійних булевих функціональних перетворень широко використовується в сучасних алгоритмах захисту інформації. При цьому проблема захищеності цих алгоритмів від злому є тотожною рішенню еквівалентної системи нелінійних булевих рівнянь. За останні два десятиліття запропоновано ряд підходів до скорочення перебору при вирішенні зазначеної математичної задачі. Найбільш відомими з них є лінійний і диференційний криптоаналіз. Природно, що досить глибоко досліджена проблема залежності можливості скорочення перебору від властивостей булевих функцій складових перетворення.

Показано, що основними властивостями булевих перетворень, що визначають обсяг перебору для вирішення математичної задачі знаходження коренів системи булевих рівнянь є нелінійність булевих функцій і їх диференціальні характеристики.

Булева функція $f(x_1, ..., x_n)$ від n змінних визначена на 2n можливих наборах значень, що належать множині Z, і приймає значення на множині {0,1}. Хеммінгова вага (англ. Hemming weight) $W(f(x1, ..., xn))$ булевої функції $f(x_1, ..., x_n)$ від n змінних є сумарне число одиничних значень, які

приймає функція на всіх 2^n можливих наборах значень змінних, які утворюють множину Z:

$$W(f(x_1,...,x_n)) = \sum_{x_1,...,x_n \in Z} f(x_1,...,x_n). \quad (1.1)$$

Булева функція $f(x_1, ..., x_n)$ відповідає критерію максимуму повної ентропії, тобто є балансної, якщо вона з однаковою ймовірністю приймає значення нуля і одиниці:

$$W(f(x_1,...,x_n)) = 2^{n-1}. \quad (1.2)$$

Система булевих функцій $G = \{f_1(x_1, ..., x_n), f_2(x_1, ..., x_n), ..., f_n(x_1, ..., x_n)\}$ є ортогональною, якщо сума по модулю 2 (XOR) будь-якої підмножини функцій цієї системи є балансної функцією:

$$\forall \vartheta \subseteq G : W(\bigoplus_{f_j \in \vartheta} f_j(x_1,...,x_n)) = 2^{n-1}. \quad (1.3)$$

Для застосувань булевих перетворень, пов'язаних із захистом інформації, важливу роль відіграє нелінійність булевих функцій. Саме цей фактор визначає стійкість до злому алгоритмів захисту інформації методами лінійного криптоаналізу.

Нелінійність - $N(f(x_1, ..., x_n))$ булевої функції $f(x_1, ..., x_n)$ визначається як мінімальна хеммінгова відстань до лінійних функцій:

$$N(f(x_1,...,x_n)) = \min_{a_k \in \{0,1\}, k=0,...,n} W(f(x_1,...,x_n) \oplus (a_0 \oplus \bigoplus_{j=1}^{n} a_j \cdot x_j)). \quad (1.4)$$

Стосовно систем булевих функцій поняття нелінійності означає мінімальну лінійність лінійних комбінацій булевих функцій, що становлять систему $G = \{f_1(x_1, ..., x_n), f_2(x_1, ..., x_n), ..., f_n(x_1, ..., x_n)\}$:

$$N(G) = \min_{b_k \in \{0,1\}, k=1,...,n} N(\bigoplus_{k=1}^{n} b_k \cdot f_k(x_1,...,x_n)), f_k \in G. \quad (1.5)$$

Нелінійність системи G булевих функцій зазвичай визначається з використанням її лінійного профілю - спеціальної таблиці, стовпці якої

відповідають лінійним комбінаціям булевих функцій, що утворюють систему, а рядки - лінійним функціям. На перетині і-того стовпця і j-того рядка вказується хеммінгова відстань між і-тою лінійною комбінацією функцій, що складають систему, і j-тою лінійною функцією. Нелінійність системи булевих функцій відповідає мінімальному елементу лінійного профілю системи G.

Важливе значення для ефективності використання булевих перетворень в алгоритмах захисту інформації мають диференціальні характеристики булевих функцій. При використанні в алгоритмах захисту інформації диференціальні характеристики булевих функцій визначають захищеність цих алгоритмів від зломів методами диференціального криптонаналізу і виключення змінних. Найважливішою диференціальною характеристикою булевих функцій є диференціальна ентропія. Диференціальна характеристика показує як змінюється булева функція при зміні вхідних змінних на яких визначена ця функція.

Диференціальна ентропія булевої функції $f(x_1, ..., x_n)$ визначається ймовірністю P_k зміни значення функції при зміні k вхідних змінних з n, на яких визначена ця функція. Для хеш-адресації і алгоритмів захисту інформації найбільше значення мають булеві функції, мають максимальну диференціальної ентропією, тобто функції, що змінюють свої значення з імовірністю 0.5 при зміні будь-яких k вхідних змінних. Такі функції отримали спеціальну назву функцій, що задовольняють критерію поширення по k змінним або просто PC (k) -функцій (Propagation Criterion on k variables або скорочено PC(k)). Формально, булева функція $f(x_1, ..., x_n)$ відповідає критерію максимуму диференціальної ентропії по k змінних або -PC (k) - функцій, якщо виконується умова:

$$W(f(X) \oplus f(X \oplus A)) = 2^{n-1}, A = \{a_1,...,a_n\}, a_j \in \{0,1\}, \sum_{j=1}^{n} a_j \le k. \qquad (1.6)$$

Особливе значення мають булеві функції, що задовольняють критерію максимуму диференціальної ентропії по одній змінній. Такі функції отримали спеціальну назву функцій, що володіють лавинним ефектом - Strict Avalanche Effect.

Булева функція $f(x_1, ..., x_n)$ відповідає критерію максимуму умовної ентропії по одній змінній або Strict Avalanche Criterion (SAC), якщо зміна будь-якої з її n змінних має наслідком зміну значення функції з ймовірністю 0.5:

$$\forall j \in \{1,..,n\} : W(f(x_1,...,x_j,...,x_n) \oplus f(x_1,...,\bar{x}_j,...,x_n)) = 2^{n-1}. \quad (1.7)$$

При k = n PC(n) функція, яка змінює своє вихідне значення з імовірністю 0.5 при інверсії всіх n вхідних змінних називається bent-функцією. Доведено, що bent-функція має максимально можливе значення нелінійності, але принципово не може бути балансною. Поняття bent-функції може бути визначено тільки для парних значень n, і її нелінійність дорівнює:

$$N(f_b(x_1,...,x_n)) = 2^{n-1} - 2^{n/2-1}. \quad (1.8)$$

Для балансних булевих функцій $f(x_1, ..., x_n)$ від n змінних значення нелінійності N(f) за умови, що n> 3 обмежена зверху межами:

$$N(f) \le 2^{n-1} - 2^{n/2-1} - 2 \text{ для парних n, и} \quad (1.9)$$

$$N(f) \le \lfloor 2^{n-1} - 2^{n/2-1} \rfloor \text{ для непарних n,}$$

причому $\lfloor x \rfloor$ - означає максимальне ціле, менше або рівне x.

Для реалізації ідентифікації відповідно до концепції "нульового знання" з використанням необоротних булевих функціональних перетворень, пропонується в якості інформації, що надається абонентом А системі (аналог "відкритого ключа"), використовувати опис самого перетворення F(X) в процедурній формі і значення вихідного вектора U_A .

Вхідний вектор X, для якого перетворення F(X) формує вихідний вектор U_A, є секретним разовим "паролем" користувача А. Незворотність

перетворення F(X) гарантує, що практично неможливо відновити вхідний вектор X за наявними в системі функціональному перетворенню F(X) і вектору U_A.

Однак така схема придатна тільки для одного циклу ідентифікації. Для того, щоб реалізувати багаторазову ідентифікацію, при якій "пароль" змінювався б при кожному зверненні абонента А до системи, необхідно, щоб необоротне функціональне перетворення F(X) мало властивістю неоднозначності, тобто існувала множина $\Omega A = \{X_1, X_2, \ldots, X_g\}$, що складається з g вхідних векторів, для кожного з яких виконується умова: $\forall X \in \Omega: F(X) = U_A$.

Для того, щоб використовувати в схемі ідентифікації на основі концепції нульового знання булеві функціональні перетворення, що володіють властивістю неоднозначності, необхідно розробити спеціальні методи їх отримання в процедурній формі.

Пропонований спосіб ідентифікації віддалених абонентів багатокористувацьких систем передбачає наявність такого методу у кожного із зареєстрованих абонентів багатокористувацької системи. Тоді на етапі реєстрації абонента А спосіб передбачає здійснення наступної послідовності дій:

1. Абонент А посилає системі запит на реєстрацію.

2. Система посилає абоненту А відкритий код програми формування відкритої частини ідентифікаційної інформації (функціонального перетворення $F_A(X)$ і вихідного коду U_A).

Абонент формує або вибирає вхідні ідентифікучі коди, що становлять множину ΩА.

4. З використанням отриманої програми для множини ΩА абонетов формує булевого функціональне перетворення $F_A(X)$ в процедурній формі, таке, щоб для кожного з ідентифікуючих його кодів множини ΩА виконувалася умова: $\forall X \in \Omega A: F_A(X) = U_A$.

5. Абонент відсилає опис отриманого в пп.4 перетворення $F_A(X)$ і вихідного коду U_A в систему. Зазначені коди є відкритою частиною ідентифікаційної системи. Вони зберігаються в системі в області пам'яті, яка відноситься до користувача А.

При виконанні *l*-го звернення до системи *($l \in \{1, ..., g\}$)* абонентом А виконується наступна послідовність дій:

1. Користувач А вибирає *l*-тий код X_l з множини Ω і відсилає його разом зі своїм номером N_A в систему.

2. Система зчитує за номером N_A абонента А з області пам'яті опис функціонального перетворення $F_A(X)$ користувача А та його код U_A.

3. Система обчислює значення коду $Y = F_A(X_l)$ для надійшовшого від абонента А коду X_l і здійснює порівняння отриманого коду Y з кодом U_A, що зберігається в пам'яті. Якщо зазначені коди рівні, тобто: $Y = U_A$, то користувачеві А надається доступ до ресурсів системи. Перетворення $F_A(X)$ модифікується системою таким чином, щоб виключити використання в подальшому коду X_l в якості ідентифікуючого коду користувача А. Якщо $Y = U_A$, то користувачеві А відмовлено в праві доступу.

Схематично, запропонований спосіб ідентифікації віддалених абонентів, заснований на концепції нульового знання і відрізняються використанням для його реалізації булевих функціональних перетворень замість модулярних операцій над числами великої розрядності показаний на рис.2.1.

Цілком очевидно, що запропонований спосіб ідентифікації укладається в рамки теоретичної концепції "нульового знання", оскільки:

- системі не відомий ідентифікуючий код користувача А для кожного сеансу до його використання абонентом і за наявною в системі інформації не існує практичної можливості дізнатися цей код до моменту його використання абонентом;

- система має можливість перевірити по коду абонента, що надійшов, його права доступу до ресурсів системи;

- після сеансу ідентифікації використаний в його рамках код ідентифікації абонента блокується, що виключає його повторне використання;

- в рамках кожного сеансу ідентифікації абонента використовується новий ідентифікуючий код.

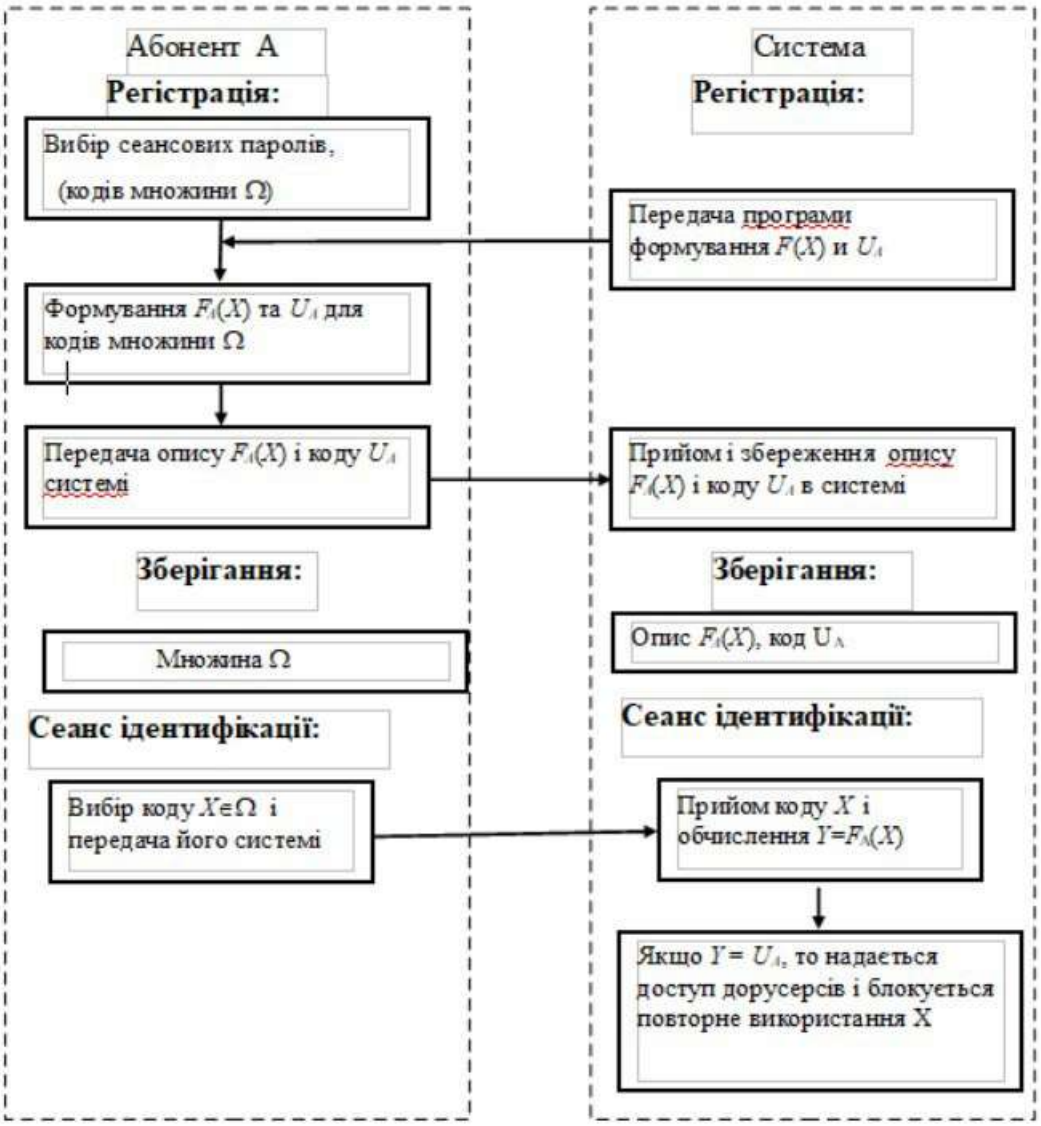


Рис.2.1. Схема ідентифікації абонентів на основі концепції "нульового знання" з використанням булевих функціональних перетворень.

Для практичної реалізації запропонованого способу ідентифікації абонентів, заснованого на концепції "нульового знання" необхідно розробити метод отримання булевих функціональних перетворень, що володіють

зазначеними вище властивостями незворотності і неоднозначності, а також структуру процедури його обчислення (процедурну форму).

2.2 Розробка і опис структури

Нехай на вхід нам подається деяке повідомлення. Воно розбивається на окремі блоки по 128 біт. Позначимо кількість блоків як m. Тоді в нашій системі в кожному раунді буде по m фрагментів, кожен з яких перетворює свій блок інформації.

На вхід фрагмента подається блок інформації W_{ij}, де i — порядковий номер раунда системи, j – порядковий номер фрагмента у раунді (рис 2.3). Функція перетворює цей блок наступним чином: вхідний вектор позначається адресою у відповідній таблиці(рис 2.2), а на вихід подається блок інформації V_{ij}, що знаходиться за адресою W_{ij} в цій в таблиці.

Надалі виконується операція xor над V_{ij} та W_{ij+1}або V_{ij} та W_{i0} , якщо j = m. Результат цієї операції подається на вхід фрагмента наступного рівня.

Алгоритм виконується задану кількість рівнів.

Підсумковим значенням буде об'єднання результатів усіх фрагментів. Результат за довжиною буде таким самим, як і вхідне повідомлення.

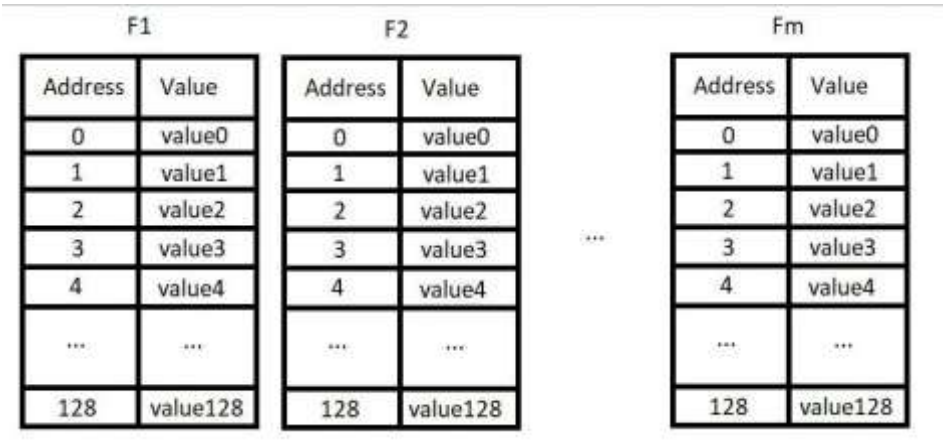


Рис 2.2 Таблиці для m функцій

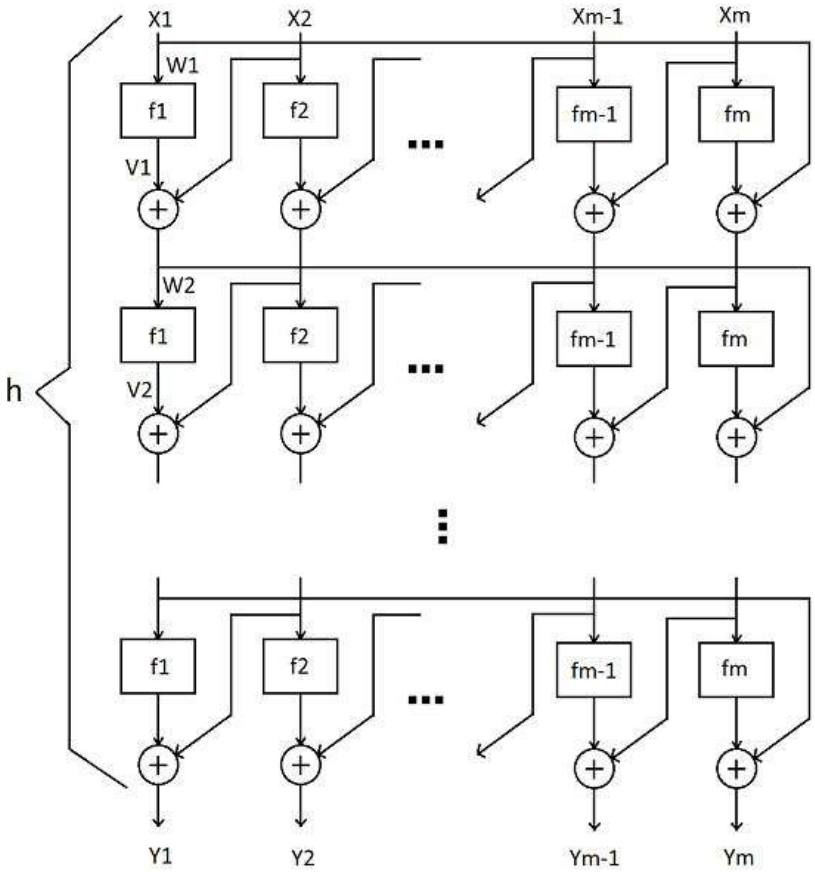


Рис. 2.3 Структура алгоритму

2.3 Створення алгоритму формування таблиць функціонального перетворення

Розроблений алгоритм реалізує формування таблиць функціонального перетворення користувачем, побудований на ідеї рекурсивного проходження рівнів системи доки не буде пройдена задана кількість рівнів.

1.Проходження одного раунда системи(рис. 3.3):

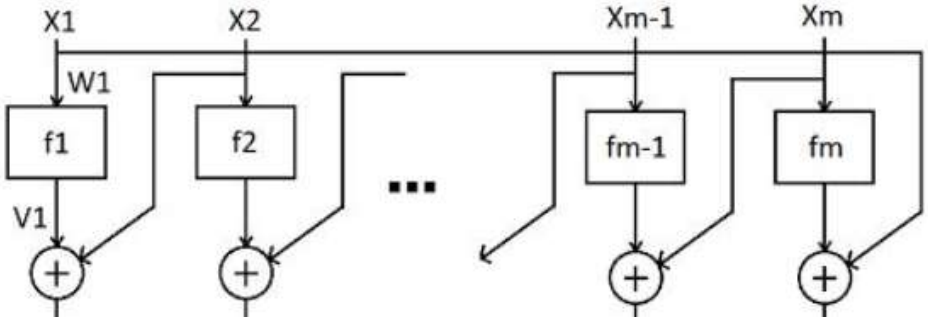


Рис 3.3 Один раунд системи

1.1. Створюється таблиця комбінацій a xor b = yi (паралельно для кожної з fi) шляхом знаходження пари для чисел від 0 до N-1, де N – кількість елементів вхідного вектора.

1.2.Комбінації в кожній таблиці перемішуться між собою.

1.3.Зі створених таблиць рекурсивно збирається нова таблиця(TABLE_A), кожен елемент якої - з N комбінацій. Елемент цієї таблиці має вигляд такого списку:(a1,b1), (a2,b2), … (an,bn). Кількість елементів таблиці — по 2^R. R – розрядність елементів вхідного вектора(наступні генеруються в разі, якщо ці елементи не підійдуть).

1.4.Проходячись по елементам нової таблиці, ми дивимось, чи задовільняє список комбінацій умові “другий елемент комбінації має бути адресою першого елемента наступної(у випадку останньої комбінації в списку — нульової) у відповідній таблиці”. Другою умовою є те, що запис не повинен суперечити вже внесеним в таблицю даним.

1.4.1. Коли ці умови виконані, ми записуємо ці комбінації у відповідні таблиці.

1.4.2 Якщо умови не виконуються, ми переходимо до наступного списку комбінацій. (Якщо списки закінчились, генеруємо наступну TABLE_A. Якщо вичерпані всі варіанти і наступну таблицю ми одержати не можемо, генерування вихідного вектору x1x2x3...xn визнається неможливим і завершується). Якщо запис був внесений в таблицю, формуємо новий вхідний вектор з адрес перших елементів комбінацій і передаємо його для проходження наступного рівня, якщо ні — переходимо до наступного списку комбінацій в таблиці.

2. Незаповнені значення в таблицях заповнюються випадковими значеннями з тих, що залишились невикористаними.

ВИСНОВКИ ДО РОЗДІЛУ 2

В даному розділі було розроблено спосіб ідентифікації абонентів на основі концепції "нульових знань" з використанням незворотних булевих функціональних перетворень, що володіють зазначеними властивостями незворотності і неоднозначності, а також структуру процедури їх обчислення (процедурну форму).

Основною перевагою запропонованого способу є те, що для реалізації концепції нульового знання використовуються булеві перетворення, реалізація яких має суттєво меншу обчислювальну складність в порівнянні мультиплікативними операціями модулярної арифметики, виконуваними над багаторозрядними числами. Це дозволяє помітно підвищити продуктивність ідентифікації в порівнянні з відомими схемами. Іншою важливою перевагою запропонованого способу є те, що він використовує, на відміну від відомих реалізацій ідентифікації, заснованих на концепції "нульового знання" тільки одну посилку по каналу передачі даних, що, з одного боку, дозволяє підвищити швидкість ідентифікації, а з іншого, мінімізувати використання лінії передачі даних - потенційно найбільш небезпечного для спроб незаконного доступу елементів схеми ідентифікації.

Недоліком способу є те, що число вхідних кодів, що ідентифікують кожного з користувачів і складають множину Ω обмежена. Другим недоліком є відносно великий обсяг інформації, що вимагається для опису сформованого користувачем булевого незворотного функціонального перетворення F(X).

РОЗДІЛ 3.

ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАПРОПОНОВАНОГО АЛГОРИТМУ

3.1. Обґрунтування вибору мови програмування

Для розробки програмного додатку було вирішено використовувати мову програмування C#[8], оскільки однією з її переваг є швидкість. Мова C# була створена компанією Microsoft спеціально для використання в операціних системах сімейства Windows для CLR[8] – Commom Language Runtime - “загального середовища виконання мов” - компоненту пакету Microsoft .NET Framework, віртуальної машини, на якій виконуються всі мови програмування .NET Framework. Тому ця мова має можливості для коректного виконання програм на Windows.

Також слід зазначити, що мова програмування C# з'явилася відносно нещодавно, вона використовується для програмування сучасних програмних додатків і систем. Технології, які дозволяє використовувати дана мова, є актуальними, а їх кількість постійно зростає, адже її розробником є одна з провідних компаній в галузі IT. Це дозволяє вибрати необхідні засоби в рамках виконання дипломного проекту та розробити програмне забезпечення, що є сучасним та легко розширюваним.

3.2 Розробка сценаріїв використання программного додатка

Запустивши програмний додаток, користувач може увійти в систему, якщо в нього є ключ, або зареєструватись. Під час реєстрації користувач формує функціональне перетворення, пароль, що зберігатимуться в системі у відкритому доступі, а також формує множину ключів, які будуть ідентифікувати користувача в системі. Діаграма прецедентів операції авторизації користувача в системі зображена на рис. 3.1.

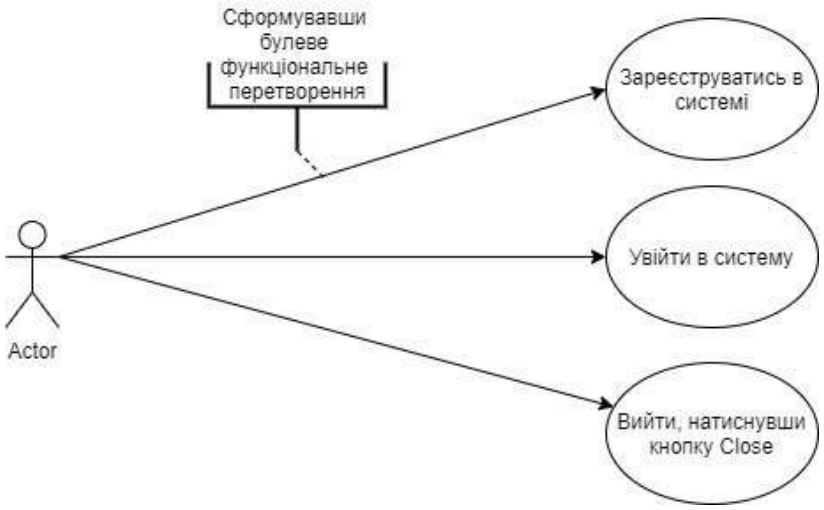


Рис 3.1. Діаграма прецедентів операції авторизації користувача в системі.

Після запуску програми користувач може виконати одну з наступних дій:

- зареєструватись в системі, сформувавши при цьому необхідні дані за алгоритмом, розробленим в ході дипломного проекту;

- увійти в систему;

- закрити програмний додаток.

Розглянемо сценарій, коли користувач реєструється. Назва: реєстрація користувача.

Учасники: користувач, система.

Передумови: користувач натискає кнопку «зареєструватись».

Результат: сформовано функціональне перетворення, визначений перетворений пароль та множина ключів, виконується реєстрація користувача в системі.

Основний сценарій:

1) користувач натискає кнопку “Register”;

2) користувач вводить необхідні дані;

3) система генерує функціональне перетворення за розробленим алгоритмом;

4) користувач отримує множину ключів у полі результату, час, витрачений системою - у полі таймера, згенеровані алгоритмом функціональні перетворення і пароль зберігаються у системі.

Розглянемо сценарій, коли користувач авторизується. Назва: ідентифікація віддаленого користувача.

Учасники: користувач, система.

Передумови: користувач вводить логін і пароль у відповідні поля та натискає кнопку “Sign In”.

Результат: виконується ідентифікація віддаленого абонента системою.

Основний сценарій:

1) користувач вводить логін і пароль у відповідні поля;

2) користувач натискає кнопку “Sign In”;

3) система перевіряє, чи підходить пароль, пропускаючи його через відповідне функціональне перетворення;

4) система допускає користувача до системи, блокуючи використаний ключ, або не допускає його вхід.

3.3 Обґрунтування вибору програмних засобів, технологій, допоміжних бібліотек, застосованих при розробці програмного забезпечення

Оскільки було вирішено розробляти програмний додаток на мові C# для виконання у операціних системах Windows, для графічного інтерфейсу використаємо технологію WPF(Windows Presentation Foundation)[7].

Серед переваг даної технології можна виділити такі:

- вона новіша за інші і, відповідно, відповідає сучасним стандартам розробки;

- Microsoft використовує її в багатьох своїх додатках, наприклад Visual Studio;

- це більш гнучка система, Ви можете зробити більше, без написання або покупки готових елементів управління;

-якщо раптом Ви вирішите скористатися готовими рішеннями із товариства, швидше за все сторонні розробники будуть сфокусовані саме на WPF, оскільки це більш нова система;

- за допомогою XAML[7] можна легко створювати і редагувати GUI, дозволяючи розділити роботу дизайнера (XAML) і програміста (C #, VB.NET і ін.);

- прив'язка даних дозволяє Вам ще краще розділити дані і GUI;

- для кращої продуктивності, можна використовувати апаратне прискорення при відображенні GUI, що дозволяє створювати GUI як для Windows додатків, так і для додатків Web (Silverlight / XBAP).

Вибір шаблонів проектування:

На сьогоднішній день окрім шаблону MVC існує безліч його модифікацій,які мають свої переваги і недоліки. Метою даного підпункту є ознайомлення з найбільш поширеними модифікаціями цього патерну.

Шаблон Model View Controller (MVC) визначає, що програма складається з моделі даних, інформації про презентацію та інформації управління. Шаблон вимагає, щоб кожне з них було розділене на різні об'єкти. MVC є скоріше архітектурним шаблоном, але не для всього програмного додатка. MVC здебільшого відноситься до інтерфейсу користувача / шару взаємодії програми. Вам все одно знадобиться рівень бізнес-логіки, можливо, якийсь рівень обслуговування та рівень доступу до даних. Концепція MVC була описана в 1979 році Трюгве Реенскаугом, який на той час займався розробкою Smalltalk в Xerox PARC. Оригінальна реалізація описана в статті «Applications Programming in Smalltalk-80: How to use Model- View-Controller». Потім Джим Алтофф з командою розробників реалізували версію MVC для бібліотеки класів Smalltalk-80.

В початковій концепції була описана роль кожного з елементів: моделі, відображення і контролера. Але зв'язки між ними не були описані конкретно . Існує два різновиди даного шаблону:

1. Пасивна модель. Модель жодним чином не впливає на контролер чи відображення, і використовується ними як джерело даних для відображення. У пасивній моделі Контролер є єдиним класом, який маніпулює моделлю. Виходячи з дій користувача, Контролер повинен змінити Модель. Після оновлення моделі Контролер повідомить Відображення, що його також потрібно оновити. У цей момент Відображення запитає дані від Моделі.

2. Активна модель. У випадках, коли Контролер - не єдиний клас, який модифікує Модель, Моделі потрібен спосіб повідомляти Відображення та інші класи про оновлення. Це досягається за допомогою шаблону спостерігача. Модель містить колекцію спостерігачів, які цікавляться оновленнями. Відображення реалізує інтерфейс спостерігача та реєструється як спостерігач до моделі. Кожен раз, коли модель оновлюється, вона також повторюватиметься шляхом збирання спостерігачів та виклику методу оновлення. Реалізація цього методу у Відображенні призведе до запиту останніх даних із Моделі.

За класичну реалізацію концепції MVC вважають варіант саме з активною моделлю. В основі класичної моделі лежать такі принципи:

1. Слабке зв'язування.

2. Модель нічого не знає ні про кого. Модель розсилає оповіщення, які може слухати, наприклад, відображення, якщо хоче.

3. Відображення знає про модель, але не може її змінювати, відображення може маніпулювати контролером.

4. Контролер знає про Модель і може її змінювати, а також знає про відображення і може його (їх) змінювати. Концепція MVC дозволяє розділити дані, подання та обробку дій користувача на три окремих компоненти:

1. Модель. Модель містить лише чисті дані програми, вона не містить логіки, що описує, як представити дані користувачеві.

2. Відображення, подання. Відображення представляє користувачеві дані моделі. Подання знає, як отримати доступ до даних моделі, але не знає, що ці дані означають або що може зробити користувач, щоб маніпулювати ними.

3. Контролер. Контролер знаходиться між видом і моделлю. Він слухає події, викликані поданням (або іншим зовнішнім джерелом), і виконує відповідну реакцію на ці події. У більшості випадків реакція викликає метод на моделі. Оскільки подання та модель пов'язані через механізм сповіщення, результат цієї дії автоматично відображається у поданні.

Важливо відзначити , що як відображення , так і контролер залежать від моделі. Однак модель не залежить ні від відображення, ні від контролера. Тим самим досягається призначення такого поділу: він дозволяє будувати модель незалежно від візуального представлення, а також створювати кілька різних відображень для однієї моделі .

Для реалізації схеми Model-View-Controller використовується досить велика кількість шаблонів проектування (залежно від складності архітектурного рішення) , основні з яких « спостерігач », «стратегія», « компонувальник »[9]. Найбільш типова реалізація відокремлює відображення від моделі шляхом встановлення між ними протоколу взаємодії, використовуючи апарат подій (підписка або сповіщення). При кожній зміні внутрішніх даних в моделі вона оповіщає всі залежні від неї відображення, і відображення оновлюється. Для цього використовується шаблон «спостерігач». При обробці реакції користувача відображення обирає, залежно від потрібної реакції, потрібний контролер , який забезпечить той чи інший зв'язок з моделлю. Для цього використовується шаблон «стратегія» , або замість цього може бути модифікація з використанням шаблону «команда». А для можливості однотипного поводження з підоб'єктами складно-складеного ієрархічного виду може використовуватися шаблон

«компонувальник». Крім того, можуть використовуватися й інші шаблони проектування, наприклад, «фабричний метод», який дозволить задати за замовчуванням тип контролера для відповідного виду.

З роками об'єктно-орієнтоване програмування та поняття про шаблони проектування розвивалося все більше. На тепер створений ряд модифікацій концепції MVC, які при реалізації у різних авторів можуть відрізнятися від оригінальної .

Шаблон Model-View-View-Model (MVVM)[7] - застосовується при проектуванні архітектури програми. Спочатку був представлений співтовариству Джоном Госсманом в 2005 році як модифікація шаблону Presentation Model. Більшою своєю частиною він базується на MVC. MVVM орієнтований на сучасні платформи розробки, такі як Windows Presentation Foundation, Silverlight від компанії Microsoft, ZK framework[6]. MVVM використовується для розділення моделі та її відображення, що необхідно для їхніх змін окремо один від одного. MVVM зручно використовувати замість класичного MVC і йому подібних в тих випадках, коли в платформі, на якій ведеться розробка, присутнє «зв'язування даних». У шаблонах проектування MVC/MVP зміни в інтерфейсі не впливають безпосередньо на Модель, а попередньо йдуть через Контролер або Presenter. У таких технологіях як WPF і Silverlight є концепція «зв'язування даних», що дозволяє пов'язувати дані з візуальними елементами в обидві сторони. Отже , при використанні цього прийому застосування моделі MVC стає вкрай незручним через те, що прив'язка даних до подання безпосередньо не вкладається в концепцію MVC/MVP. Шаблон MVVM ділиться на три частини:

1. Модель, так само, як у класичному MVC, являє собою фундаментальні дані, необхідні для роботи програми .

2. Представлення - це графічний інтерфейс, тобто вікно, кнопки і т.п. Відображення є підписником на подію зміни значень властивостей або

команд, що надаються Моделлю подання. У разі, якщо в Моделі відображення змінилась якась властивість, то вона сповіщає всіх підписників про це, і Відображення, в свою чергу, запрошує оновлене значення властивості з Моделі представлення. У випадку, якщо користувач впливає на якийсь елемент інтерфейсу, Відображення викликає відповідну команду, надану Моделлю подання.

3. Модель відображення (англ. View Model) є , з одного боку , абстракцією Відображення, а з іншого, надає обгортку даних з Моделі, які підлягають скріпленню. Тобто, вона містить Модель, яка перетворена до Відображення, а також містить у собі команди, якими може користуватися Відображення, щоб впливати на Модель .

Оскільки було обрано технологію WPF для розробки графічного інтерфейсу, вирішено обрати MVVM для проектування архітектури программного дадатка.

Технології, пов'язані з базами даних:

SQL Server[11,12,13,14] є однією з найбільш популярних систем управління базами даних у світі. Саме тому її використовують всюди: від малих проектів до великих і навантажених. SQL Server був розроблений компанією Microsoft, у 1987 році перша його версія побачила світ. З тих пір минуло немало часу і технологія була покращена. На сьогодні останньою версією продукту є 16-та. З цієї версії система є доступною не лише на Windows, а також підтримує платформу Linux.

SQL Server не просто так є таким популярним серед розробників. Він має такі особливості як продуктивність, простота, а також надійність та безпека. Він працює дуже швидко та надає шифрування даних. За допомогою SQL Server досить легко працювати з базами даних.

Загалом, коли ми кажемо про системи управління базами даних, в центрі уваги самі бази даних. За допомогою БД дані зберігаються певним чином. Досить часто можна стикнутися з тим, що фізично база даних являє собою

певний файл на жорсткому диску, хоча це і не є обов’язковим. Для адміністрування баз даних застособують системи управління базами даних (англ. database management system) або СУБД (DBMS). MS SQL Server є однією з таких систем.

Для організації баз даних MS SQL Server використовує реляційну модель, що була розроблена у 1970 році Едгаром Коддом. На сьогоднішній день така модель дуже розповсюджена і застосовується майже всюди. Така модель передбачає зберігання даних у вигляді таблиць. Така таблиця поділяється на рядки та стовпці. В рядку розташований сам об'єкт, а стовпці визначають певні властивості цього об'єкта.

Для ідентифікації кожного рядка використовується первинний ключ (англ. primary key)[12]. Цей ключ може складатися з одного чи декількох стовпців. Зазвичай для данного ключа використовують стовпець id, адже він є унікальним дл будь-якого об'єкта. Це, ймовірно, класичний варіант встановлення первинного ключа. Проте, іноді можно обійтись зовсім без нього. Для цього беруться кілька стовпців з властивостями об'єкта і разом стають первинним ключем. Такий сценарій дехто вважає більш правильним адмініструванням баз даних. Використовуючи первинний ключ, ми можемо зсилатися на певний об’єкт в таблиці, а отже два об’єкти не можуть мати один і той самий первинний ключ.

Однак, існує не лише первинний ключ. В разі, коли необхідно зіслатися на іншу таблицю в базі данних, використовується зовнішній ключ (англ. foreign key)[12]. За допомогою даного ключа атрибут об'єкта однієї таблиці пов'язується з первинним ключем іншої. Сама база даних тоді стає реляційною від англійського слова relation.

Якщо в базі даних є таблиці, то є і засоби взаємодії з ними. Основним таким засобом є мова SQL (англ. Structured Query Language)[12]. Вона дозволяє користувачу, а це може бути певна зовнішня программа,

сформувати певний запит в базу даних, СУБД певним чином опрацьовує цей запит, і користувач отримує результат.

Спочатку мова SQL був розроблений в компанії IBM для системи баз даних, яка називалася System / R. При цьому сама мова називався SEQUEL (Structured English Query Language). Хоча в підсумку ні база даних, ні сам язик не були згодом офіційно опубліковані, за традицією сам термін SQL нерідко вимовляють як "сиквел".

У 1979 році компанія Relational Software Inc. розробила першу систему управління баз даних, яка називалася Oracle і яка використовувала мову SQL. У зв'язку з успіхом даного продукту компанія була перейменована в Oracle.

Згодом стали з'являтися інші системи баз даних, які використовували SQL. У підсумку в 1989 році Американський Національний Інститут Стандартів (ANSI) кодифікував мову і опублікував його перший стандарт. Після цього стандарт періодично оновлювався і доповнювався. Останнє його оновлення відбулося в 2011 році. Але незважаючи на наявність стандарту нерідко виробники СУБД використовують свої власні реалізації мови SQL, які трохи відрізняються один від одного.

Виділяються два різновиди мови SQL: PL-SQL і T-SQL. PL-SQL використовується в таких СУБД як Oracle і MySQL. T-SQL (Transact-SQL) застосовується в SQL Server. Власне тому в рамках поточного керівництва буде розглядатися саме T-SQL.

Залежно від завдання, яку виконує команда T-SQL, він може належати до одного з наступних типів:

DDL (Data Definition Language / Мова визначення даних). До цього типу належать різні команди, які створюють базу даних, таблиці, індекси, збережені процедури і т.д. У загальних рисах визначають дані.

Зокрема, до цього типу ми можемо віднести наступні команди:

- CREATE: створює об'єкти бази даних (саму базу даних, таблиці, індекси і т.д.);

- ALTER: змінює об'єкти бази даних;

- DROP: видаляє об'єкти бази даних;

- TRUNCATE: видаляє всі дані з таблиць.

DML (Data Manipulation Language / Мова маніпуляції даними). До цього типу відносять команди на вибір даних, їх оновлення, додавання, видалення - в загальному все ті команди, за допомогою яких ми можемо управляти даними.

До цього типу належать такі команди:

- SELECT: витягує дані з БД;

- UPDATE: оновлює дані;

- INSERT: додає нові дані;

- DELETE: видаляє дані.

DCL (Data Control Language / Мова управління доступу до даних). До цього типу відносять команди, які керують правами щодо доступу до даних. Зокрема, це такі команди:

- GRANT: надає права для доступу до даних;

- REVOKE: відкликає права на доступ до даних.

Допоміжні бібліотеки:

- System.Windows.Controls

Дана бібліотека[10] зберігає дані про всі елементи керування, які програміст може використовувати в межах технології WPF. Наведемо ті, що будуть використані в нашому програмному додатку:

● Button

Об'єкт Button – це елемент управління, який реагує на введення даних користувачем від миші, клавіатури, пера або іншого пристрою введення і створює подію Click. Об'єкт Button являє собою простий, призначений для користувача компонент інтерфейсу, який може бути наповнений простим вмістом, таким як текст і може також містити складний вмісту, такий як зображення і Panel елементів управління.

● Canvas

Canvas представляє елемент керування макета, який дозволяє абсолютне встановлення положення дочірніх елементів.

● DataGrid

Елемент DataGrid управління дозволяє відображати і відсівати дані з різних джерел, наприклад з бази даних.

● Image

Елемент використовується для відображення растрових зображень в Windows Presentation Foundation (WPF) додатків.

● ListView

Елемент управління ListView надає інфраструктуру для відображення набору елементів даних в різних макетах або уявленнях.

● Panel

Panel є базовим класом для всіх елементів, що підтримують макет додатки в Windows Presentation Foundation (WPF) .

● ProgressBar

Об'єкт ProgressBar показує хід виконання операції. ProgressBar – це елемент управління, що складається зі стрічки, яка заповнюється зеленим кольором в процесі виконання операції.

● ScrollViewer

ScrollViewer елемент управління створює прокручувати область, в разі, якщо вміст може прокручуватися горизонтально або вертикально.

● StackPanel

StackPanel елемент використовується для розташування дочірніх елементів, горизонтально або вертикально.

● TextBlock

TextBlock – це елемент управління, який забезпечує гнучку підтримку тексту для WPF додатків. Елемент в основному призначений для базових сценаріїв ІП , які не вимагають більше одного абзацу тексту.

● TextBox

Елемент управління TextBox забезпечує підтримку базових текстових вводів в додатках WPF.

- Entity Framework

До .NET 3.5 розробники часто використовували для написання коду ADO.NET або Enterprise Data Access Block[15] для збереження або отримання даних програми з базової бази даних. Розробники повинні були відкрити з'єднання з базою даних, створити набір даних для отримання або подання даних до бази даних, перетворити дані з DataSet в .NET-об'єкти або навпаки, щоб застосувати бізнес-правила. Це був громіздкий і схильний до помилок процес. Корпорація Microsoft створила структуру під назвою "Entity Framework"[15] для автоматизації всіх цих заходів, пов’язаних із базою даних для вашої програми.

Entity Framework - це рамка ORM з відкритим кодом для програм .NET, що підтримуються Microsoft. Це дозволяє розробникам працювати з даними, використовуючи об’єкти класів, що належать до домену, не орієнтуючись на основні таблиці та колонки бази даних, де зберігаються ці дані. За допомогою Entity Framework розробники можуть працювати на більш високому рівні абстракції, коли вони обробляють дані, і можуть створювати та підтримувати орієнтовані на дані додатки з меншим кодом порівняно з традиційними програмами.

Офіційне визначення: "Entity Framework - це об'єктно-реляційний картограф (ORM), який дозволяє розробникам .NET працювати з базою даних за допомогою .NET-об'єктів. Це виключає необхідність більшості коду доступу до даних, який розробникам зазвичай потрібно писати. " На рис. 3.2 показано, де Entity Framework вписується у вашу програму.

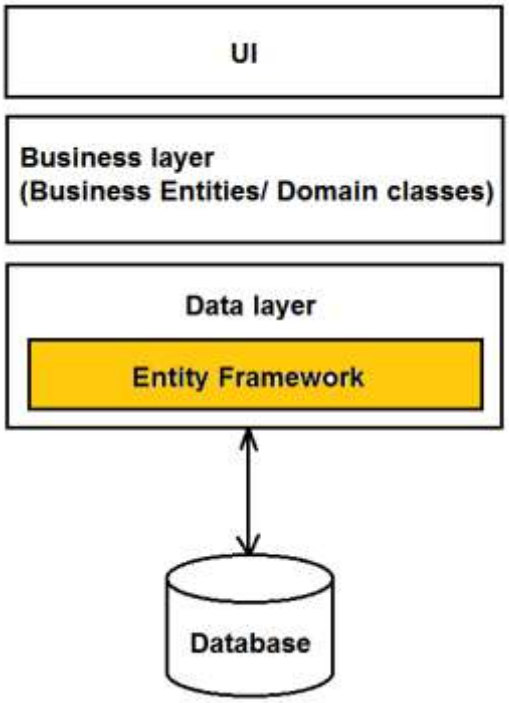


Рис 3.2 Місце Entity Framework в програмі.

Відповідно до наведеного вище рисунку, Entity Framework стає між шаром бізнес-логіки(доменних класів) та базою даних. Він зберігає дані, що зберігаються у властивостях цих класів, а також витягує дані з бази даних та автоматично перетворює їх на об'єкти доменних класів.

Особливостями Entity Framework є

1. Крос-платформеність: EF Core - це крос-платформа, яка може працювати в Windows, Linux та Mac.

2.Моделювання: EF (Entity Framework) створює EDM (модель даних особи) на основі об'єктів POCO (Plain Old CLR Object) з властивостями get / set різних типів даних. Вона використовує цю модель під час запиту або збереження даних сутності до базової бази даних.

3.Запит: EF дозволяє нам використовувати запити LINQ (C # / VB.NET) для отримання даних з базової бази даних. Постачальник баз даних переведе цей LINQ-запит на специфічну мову запитів (наприклад, SQL для реляційної

бази даних). EF також дозволяє нам виконувати необроблені SQL запити безпосередньо в базі даних.

4.Відстеження змін: EF відслідковує зміни, що відбулися у випадках, коли ваші сутності (Значення властивостей) потрібно подати до бази даних.

5.Збереження: EF виконує в базі даних команди INSERT, UPDATE та DELETE на основі змін, які відбулися у ваших об'єктах під час виклику методу SaveChanges(). EF також забезпечує асинхронний метод SaveChangesAsync().

6.Паралельність: EF використовує оптимістичну паралельність за замовчуванням, щоб захистити зміни, внесені іншим користувачем з моменту отримання даних із бази даних.

7.Операції: EF виконує автоматичне управління транзакціями під час запитів або збереження даних. Він також пропонує варіанти налаштування управління транзакціями.

8.Кешування: EF включає перший рівень кешування з коробки. Отже, повторний запит поверне дані з кешу, а не звернення до бази даних.

9.Вбудовані конвенції: EF дотримується конвенцій щодо схеми програмування конфігурації та включає набір правил за замовчуванням, які автоматично налаштовують модель EF.

10.Конфігурації: EF дозволяє нам налаштувати модель EF за допомогою атрибутів анотації даних або Fluent API для зміни умовних умов.

11.Міграції: EF надає набір команд міграції, які можна виконати на консолі диспетчера пакетів NuGet або інтерфейсі командного рядка для створення або управління базовою схемою бази даних.

Останні версії Entity Framework

Microsoft представила Entity Framework у 2008 році разом із .NET Framework 3.5. З того часу він випустив багато версій Entity Framework. В даний час існує дві останні версії Entity Framework: EF 6 та EF Core.

3.4 Розробка графічного інтерфейсу

На рис. 3.3 зображено ескіз моделі графічного інтерфейсу головного вікна клієнтської частини програмного додатка. В даному вікні користувач може ввести свій логін та пароль та натиснути кнопку "Sign In", а програмний додаток пропустить пароль через збережене в системі функціональне перетворення і надасть користувачу доступ до системи, якщо перетворений ключ і збережений системою співпадуть. Також в даному вікні можна розпочати реєстрацію в системі, натиснувши кнопку "Register".

Рис 3.3 Ескіз головного меню клієнтської частини программного додатка.

На рис. 3.4 зображено ескіз моделі графічного інтерфейсу вікна реєстрації клієнтської частини програмного додатка. В даному вікні користувач може зареєструватися в системі. Для цього йому слід заповнити поля з логіком та паролем та натиснути кнопку "Generate passwords". Тоді система формує функціональне перетворення та пароль, що зберігатимуться в системі, а також множину ключів для користувача.

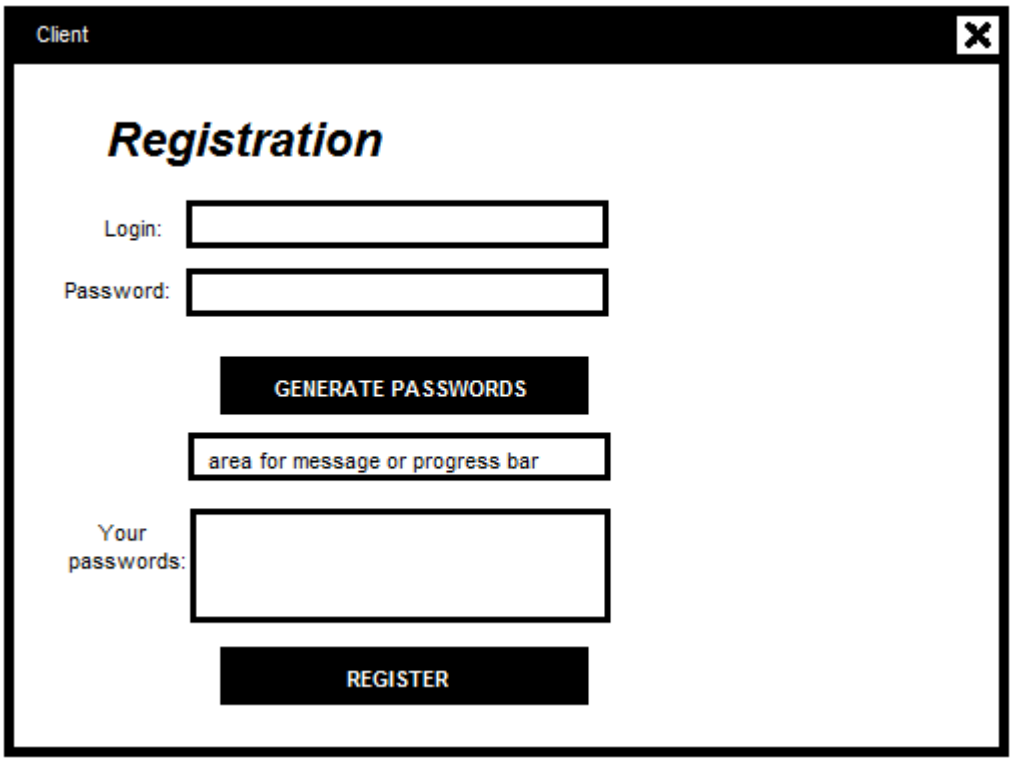


Рис 3.4 Ескіз меню реєстрації клієнтської частини программного додатка.

3.5 Розробка структури програмного додатка

Програмний додаток, що розробляється в ході дипломного проекту має дві частини: клієнтську, яка взаємодіє з користувачем, та серверну, що зберігає дані системи та виконує ідентифікацію користувача при вході в систему.

На етапі проектування було розроблено наступні класи:

ClientNet.cs - клас, що відправляє на сервер запит від клієнта на обробку даних, отримує дані з сервера та обробляє їх.

ServerNet.cs - клас, що реалізує прийом команд від клієнтів та серверну обробку даних.

Vector.cs - клас, що визначає вектор чисел.

BinaryNumber.cs - клас, що визначає двійкове число.

Table.cs - клас, що визначає таблицю.

Combination.cs - клас, що визначає пару двійкових чисел.

TableBuilder.cs - клас, що визначає дії алгоритму генерації таблиць для функціонального перетворення.

KeyFinder.cs – клас, що генерує список ключів для клієнта за задними таблицями функціональних перетворень.

IdentificationSystem.cs - клас, що реалізує метод побудови незворотних булевих перетворень для криптографічно строгої ідентифікації віддалених користувачів.

MainWindowViewModel.cs - клас, що визначає viewmodel вікна меню.

RegistrationWindowViewModel.cs - клас, що визначає viewmodel вікна реєстрації.

RelayCommand.cs — визначає базовий клас для команд

ViewModelBase.cs — визначає базовий клас для viewmodel

Спроектовану структуру программного додатка показано в ДОДАТКУ 1. Діаграма класів представлена в ДОДАТКУ 2.

3.6 Створення реляційної бази даних для серверної частини

Для серверної частини програмного додатка створимо реляційну базу даних. В рамках потреб даного дипломного проекту на серверній частині зберігається інформація про абонентів, а саме: логін користувача, перетворений пароль користувача та таблиці для булевих функціональних перетвореннь - вказуються та генеруються користувачем при проходженні процедури реєстрації. А це означає, що будуть створені наступні таблиці:

- Client зберігатиме логін та перетворений пароль абонента;
- FTable - булеве функціональне перетворення.

```
create table Client
(id int identity,
 [login] varchar(250),
 [password] varchar(250))
```

create table FTable

(id int identity,

name varchar(250),

[address] varchar(250),

value varchar(250))

Оскільки кожен користувач формує власне булеве фунціональне перетворення, необхідно встановити зв'язок між двома створеними таблицями. Цей зв'язок забезпечуватиме таблиця ClientTables.

create table ClientTable

(id int identity,

client_id int,

table_id int)

Також слід блокувати паролі, вже використовані користувачем. Такі паролі зберігатимуться в таблиці BlockedCodes.

create table BlockedCode

(id int identity,

client_id int,

code varchar(250))

Наостанок задаємо первинні(англ. primary key) та вторинні ключі(англ. foreign key).

ALTER TABLE Client ADD CONSTRAINT pk1 PRIMARY KEY NONCLUSTERED(id)

ALTER TABLE FTable ADD CONSTRAINT pk2 PRIMARY KEY NONCLUSTERED(id)

ALTER TABLE ClientTable ADD CONSTRAINT pk3 PRIMARY KEY NONCLUSTERED(id)

ALTER TABLE BlockedCode ADD CONSTRAINT pk4 PRIMARY KEY NONCLUSTERED(id)

ALTER TABLE ClientTables ADD CONSTRAINT fk1 FOREIGN KEY(client_id) REFERENCES Client(id)

ALTER TABLE ClientTables ADD CONSTRAINT fk2 FOREIGN KEY(table_id) REFERENCES FTable(id)

ALTER TABLE BlockedCodes ADD CONSTRAINT fk3 FOREIGN KEY(client_id) REFERENCES Client(id)

Отримана в результаті база даних зображена за допомогою діаграми на рис. 3.5

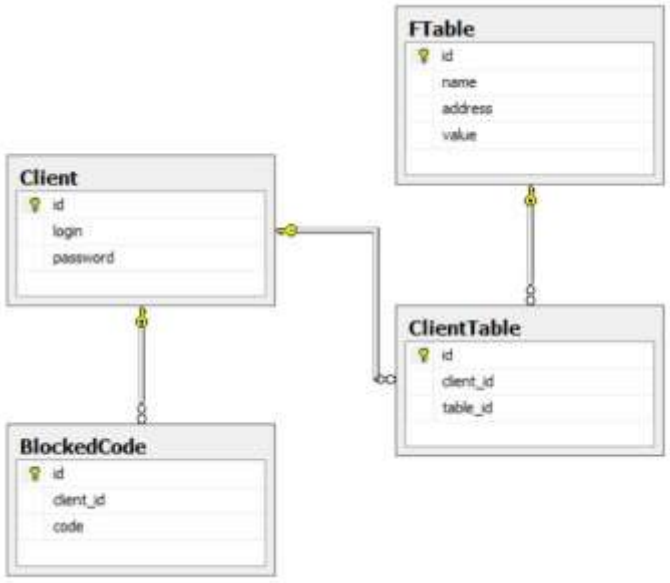


Рис 3.5. Діаграма бази даних.

3.7 Специфікації класів программного модуля, що реалізує структуру та метод ідентифікації віддаленого користувача в системі

Vector.cs

- public short[] vector { get; set; }

Масив, що зберігає вектор чисел.

- public Vector(short[] vect)

Конструктор класу.

- public static short[] strToAsciiVector(string pass)

Метод, що перетворює рядок в список кодів символів.

- public static string asciiVectorToStr(short[] vec)

Метод, що перетворює список кодів символів в рядок.

- public override string ToString()

Метод, що перевизначає конвертування екземпляру класу у рядок.

BinaryNumber.cs

- short number;

Змінна для зберігання числа в десятковій системі числення.

- public static Random random = new Random();

Змінна для генерування випадкових чисел.

- public short IntNumber { get { return number; } set { number = value; } }

Властивість, що визначає десяткове представлення числа.

- public string BinNumber { get { return Convert.ToString(number, 2); } set {number = Convert.ToSByte(value, 2); } }

Властивість, що визначає двійкове представлення числа.

- public BinaryNumber(short num)

Конструктор класу.

- public BinaryNumber(string num)

Конструктор класу.

- public static BinaryNumber getRandom(short max)

Метод, що генерує одне випадкове двійкове число.

- public static BinaryNumber[] shortVectorToBinaryVector(short[] vector)

Метод, що конвертує вектор чисел у десятковій системі числення у вектор двійкових чисел.

-public static short[] binaryVectorToShortVector(BinaryNumber[] binary_vector)

Метод, що конвертує вектор двійкових чисел у вектор чисел у десятковій системі числення.

- public override string ToString()

Метод, що перевизначає конвертування екземпляру класу у рядок.

Table.cs

- public BinaryNumber[] table { get; set; }

Масив для зберігання таблиці.

- public void fillWithRand()

Метод, що випадковим чином заповнює таблицю унікальними значеннями.

- public void print()

Метод, що друкує таблицю в консолі.

Combination.cs

- public BinaryNumber num1 { get; set; }

Властивість, що визначає перше двійкове число.

- public BinaryNumber num2 { get; set; }

Властивість, що визначає друге двійкове число.

- public Combination()

Конструктор класу.

- public Combination(BinaryNumber num1, BinaryNumber num2)

Конструктор класу для ініціалізації двома двійковими числами.

- public void createCombination(BinaryNumber result, BinaryNumber num)

Метод, що знаходить пару на основі результату та другого числа в двійковій системі числення.

- public override string ToString()

Метод, що перевизначає конвертування екземпляру класу у рядок.

KeyFinder.cs

- Table[] tables;

Масив, що представляє таблиці для функціонального перетворення.

- int count_levels;

Змінна, що визначає кількість раундів алгоритму.

- int number_of_digits;

Змінна, що визначає розрядність елементів ключа.

- public KeyFinder(Table[] tables, int count_levels, int number_of_digits)

Конструктор класу.

- public short rf(BinaryNumber x, int index)

Метод, що реалізує пошук адреси в таблиці під номером index за значенням x.

- public BinaryNumber f(BinaryNumber x, int index)

Метод, що реалізує пошук значення в таблиці під номером index за адресою x.

- public List<BinaryNumber[]> runLevel(BinaryNumber[] vector)

Метод, що реалізує пошук ключів для одного раунда в системі.

- public List<Vector> run(BinaryNumber[] vector)

Метод, що реалізує пошук ключів.

Identification.cs

- Table[] tables;

Масив, що представляє таблиці для функціонального перетворення.

- int count_levels;

Змінна, що визначає кількість раундів алгоритму.

- public Identification(Table[] tables, int count_levels)

Конструктор класу.

- public short rf(BinaryNumber x, int index)

Метод, що реалізує пошук адреси в таблиці під номером index за значенням x.

- public BinaryNumber f(BinaryNumber x, int index)

Метод, що реалізує пошук значення в таблиці під номером index за адресою x.

- public Vector runLevel(Vector vector)

Метод, що реалізує булеві функціональні перетворення для одного раунда в системі.

- public Vector run(BinaryNumber[] vector)

Метод, що реалізує булеві функціональні перетворення.

TableBuilder.cs

- Table[] tables;

Масив, що представляє таблиці для функціонального перетворення.

- int count_levels;

Змінна, що визначає кількість раундів алгоритму.

- List<Combination>[] vector_combinations;

Список, що визначає вектор комбінацій.

- List<List<Combination>> allCombinations;

Список всіх комбінацій, що можуть бути розглянуті в межах алгоритму.

- BinaryNumber[] vector_y;

Массив, що визначає вектор результату для кожного раунда алгоритму.

- List<int[]> indexes;

Список масивів індексів.

- public TableBuilder(Table[] tables, int count_levels)

Конструктор класу.

- public void shuffleCombinations()

Метод, що реалізує перемішування комбінацій двох чисел у двійковій системі числення.

- public BinaryNumber[] restoreLevel(int level, BinaryNumber[] vector_y)

Метод, що реалізує проходження одного рауда алгоритму.

- public BinaryNumber[] getAddressVector(List<Combination> combination)

Метод, що реалізує отримання вектора адрес з таблиці для заданої комбінації.

- public bool makeEntriesInTables(List<Combination> vect_of_combinations)

Метод, що реалізує перевірку комбінацій для внесення їх в таблицю та створення записів у таблицях.

- public void fillNextColumn(int level, int colIndex, List<Combination> mas)

Метод, що заповнює наступну колонку таблиці.

- public void findAllCombinations(object index)

Метод, що реалізує пошук всіх комбінацій комбінацій чисел, таких, що задовольняють умову x1^x2 = result

IdentificationSystem.cs

- int number_of_digits;

Змінна, що визначає кількість символів паролю.

- int number_of_levels;

Змінна, що визначає кількість раундів алгоритму.

- BinaryNumber[] vector;

Массив, що визначає пароль у вигляді вектора двійкових чисел ASCII-кодів символів.

- Table[] tables;

Масив, що визначає таблиці з кодами.

- public IdentificationSystem(short[] vector,int count_levels)

Конструктор класу.

- public Vector identification(short[] vect)

Метод, що реалізує булеві функціональні перетворення за допомогою вже створених таблиць (виконання заданої кількості раундів кожною з функцій).

- public List<Vector> findKeys(short[] vect)

Метод, що реалізує повне заповнення таблиць (виконання заданої кількості раундів кожною з функцій) за допомогою екземпляру класу TableBuilder.

- public void printTables()

Метод, що реалізує друк таблиць в консолі.

- public void fillAllTablesWithRand()

Метод, що реалізує дозаповнення таблиць випадковими унікальними числами в межах необхідного діапазону.

ВИСНОВКИ ДО РОЗДІЛУ 3

В даному розділі, враховуючи усі переваги та недоліки, було обрано мову програмування, якою буде написано програмний додаток до дипломного проекту — мову C#. Було обгрунтовано вибір сучасних для використання технологій та допоміжних бібліотек, що зроблять вихідний програмний додаток легко розширюваним та зручним у користуванні.

Також на основі розробленого алгоритму було створено програмний додаток, який дозволяє користувачу протестувати запропонований метод отримання булевих функціональних перетворень, що володіють властивостями незворотності і неоднозначності, пройшовши процедуру реєстрації в системі. При розробці программного додатка було використано об’єктно-орієнтований підхід та паттерни проектування. Це спростить взаємодію користувача з системою та дозволить її зручно доповнювати, розширювати в майбутньому.

РОЗДІЛ 4.

ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РОЗРОБЛЕНОГО МЕТОДУ ІДЕНТИФІКАЦІЇ ВІДДАЛЕНИХ КОРИСТУВАЧІВ

4.1 Інструкція користувача

У процесі виконання роботи був створений графічний інтерфейс для програмного додатка клієнтської частини.

Слід навести коротку інструкцію користувача для тих, хто буде користуватися додатком та досліджувати роботу запропонованого методу.

При запуску створеного програмного додатка користувач бачить форму аутентифікації(рис 4.1).

Рис 4.1. Форма аутентифікації

Користувач може ввести логін та пароль і увійти в систему. Якщо в абонента ще немає облікового запису, слід натиснути на кнопку "Register". При цьому відкриється вікно реєстрації (рис 4.2). Користувач повинен придумати та ввести необхідні дані: логін та пароль, а потів натиснути кнопку "Generate passwords". При цьому система згенерує таблиці для

булевого функціонального перетворення на основі вказаного пароля, а також множину паролів користувача (рис 4.3). У рядку з повідомленням пишеться час, за який були сформовані таблиці для булевих перетворень.

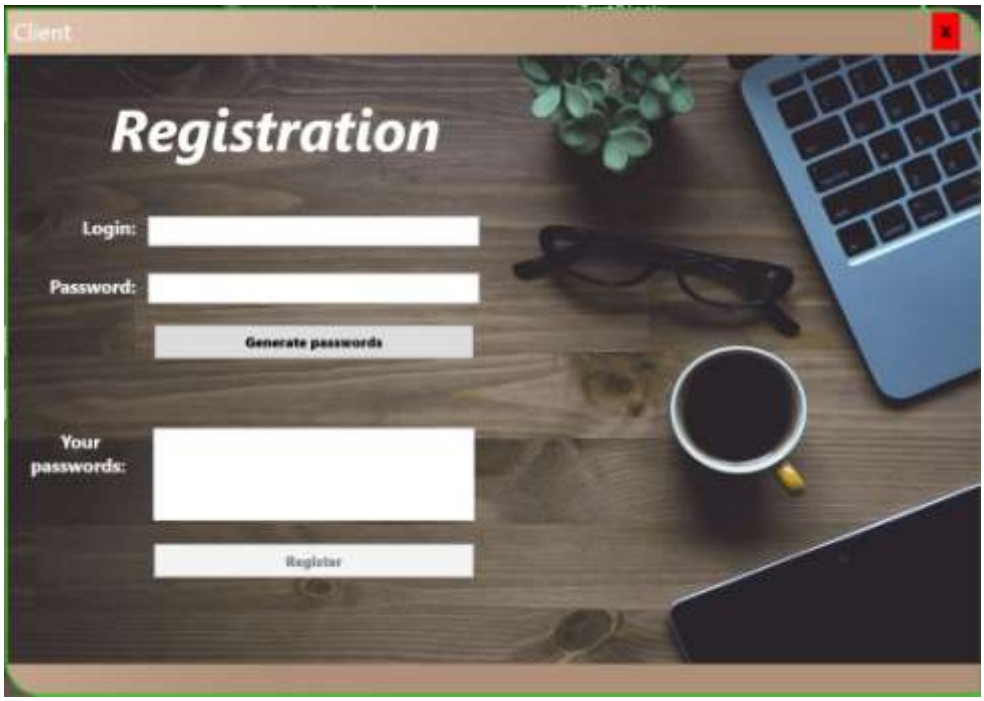


Рис 4.2 Вікно реєстрації

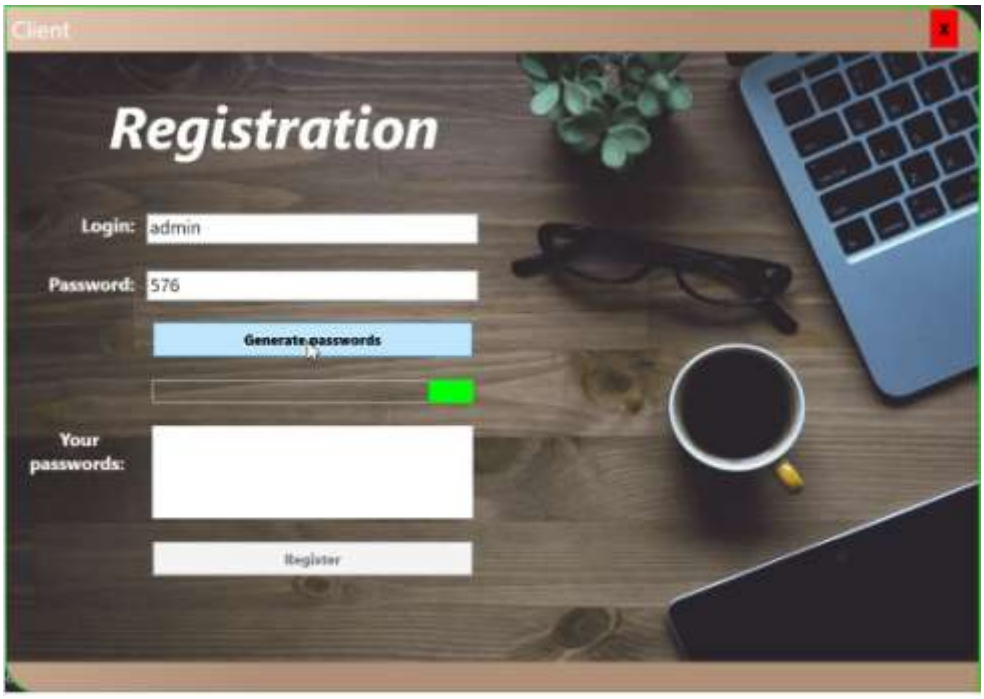


Рис 4.3 Очікування результату

Користувач отримує множину ключів для ідентифікації. Абонент підтверджує реєстрацію, натиснувши кнопку "Register" (рис 4.4) та закриває вікно.

Рис 4.4 Підтвердження реєстрації

Тепер для аутентифікаціїї користувач може ввести свій логін та один з отриманих ключів(рис. 4.5).

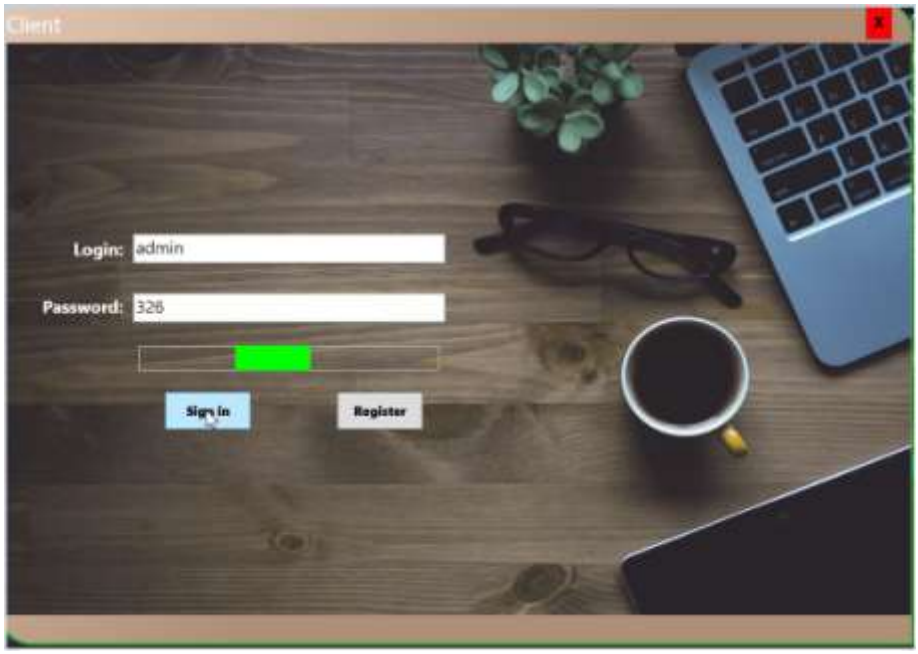


Рис 4.5 Очікування ідентифікації системою

Система ідентифікації у серверній частині програмного додатка проведе функціональні перетворення над введеним ключем та ідентифікує абонента, якщо вихідний ключ співпаде зі збереженим в базі паролем(рис 4.6). В цьому разі вона блокує цей пароль, а користувачу надає одноразовий доступ до його облікового запису.

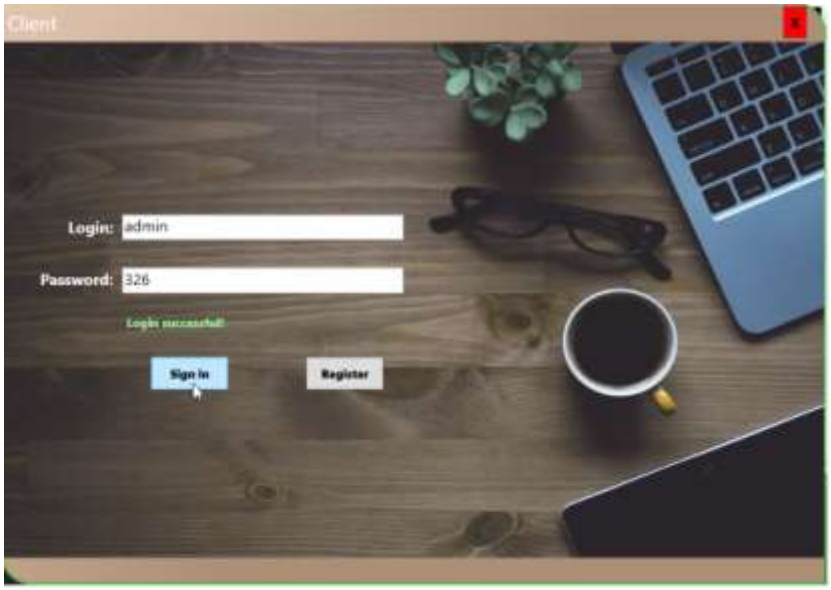


Рис 4.6 Успішна аутентифікація

В разі, якщо перетворений ключ не співпадає зі збереженим паролем,або ж, коли цей ключ виявиться вже блокованим системою, доступ користувачу до облікового запису не надається(рис. 4.7).

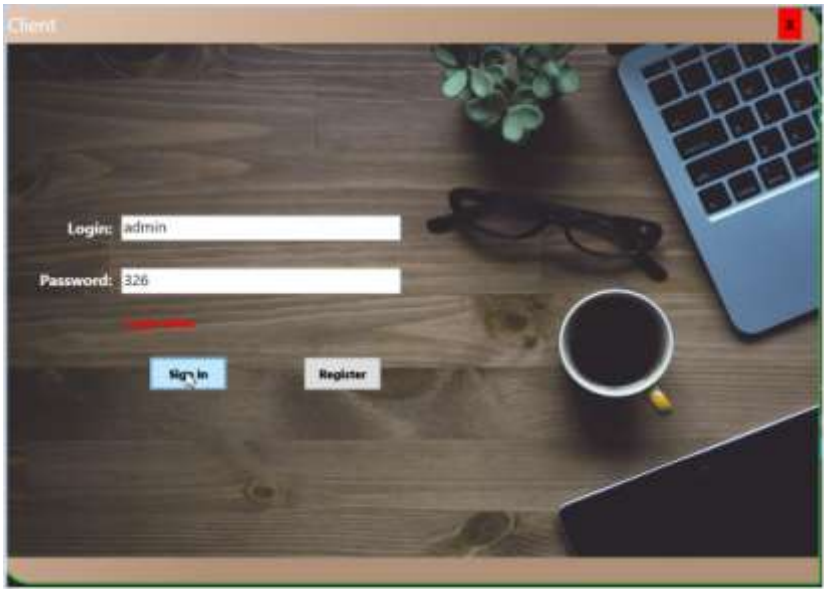


Рис 4.7 Доступ не надано

4.2 Дослідження роботи алгоритму формування таблиць для булевих перетворень

В ході виконання дипломного проекту було створено програмний модуль, що реалізує роботу розробленого алгоритму отримання булевих функціональних перетворень, що володіють властивостями незворотності і неоднозначності для ідентифікації віддалених користувачів системи. Також було створено графічний інтерфейс, що забезпечує зручні умови для дослідження роботи алгоритму.

Для дослідження роботи запропонованого методу було проведено кілька експериментів та створено таблицю швидкості генерування таблиць мулевого функціонального перетворення (табл. 4.1) в залежності від довжини паролю користувача.

Таблиця 4.1. Часова таблиця

N	Час виконання
3	Hours: 0 Minutes: 0 Seconds: 0 Milliseconds: 9
4	Hours: 0 Minutes: 0 Seconds: 0 Milliseconds: 14
5	Hours: 0 Minutes: 0 Seconds: 0 Milliseconds: 27
6	Hours: 0 Minutes: 0 Seconds: 0 Milliseconds: 407
7	Hours: 0 Minutes: 0 Seconds: 7 Milliseconds: 259
8	Hours: 0 Minutes: 2 Seconds: 33 Milliseconds: 303
9	Hours: 2 Minutes: 4 Seconds: 14 Milliseconds: 788

де N — кількість елементів входного вектора.

ВИСНОВКИ ДО РОЗДІЛУ 4

В даному розділі було розглянуто розроблений програмний додаток. Як вже було зазначено вище, він є простим і зрозумілим, легко розширюється за рахунок використаних при розробці технологій.

Використовуючи розроблений програмний додаток проведено експериментальні дослідження запропонованого методу ідентифікації віддалених користувачів.

На базі даних досліджень можна сказати, що розроблений алгоритм має суттєву перевагу: він виграє в швидкості серед інших алгоритмів при вході користувача в систему, а також наступний недолік: користувач витрачає більше часу на реєстрацію, адже йому необхідно сформувати таблиці для булевих функціональних перетворень.

ВИСНОВКИ

В ході виконання дипломного проекту було розроблено спосіб синтезу незворотних булевих функціональних перетворень для схеми ідентифікації віддалених абонентів, що реалізує крипрографічно-строгу концепцію “нульових знань”. На основі цього способу було створено алгоритм проведення булевих функціональних перетворень, систему, що працює на основі даного алгоритму, підтверджуючи його працездатність та актуальність. Також було проведено деякі дослідження швидкості роботи системи.

В результаті всіх проведених досліджень можна сказати, що розроблений алгоритм має такі переваги та недоліки:

Переваги

- використання булевих перетворень для реалізації заданої концепції дозволяє в рази прискорити процес ідентифікації в порівнянні зі схемами, що основані на нерозв'язних задачах теорії чисел і реалізація яких потребує модулярного експоненціювання над числами великої розрядності;

- він використовує, на відміну від відомих реалізацій ідентифікації, заснованих на концепції "нульового знання" тільки одну посилку по каналу передачі даних, що, з одного боку, дозволяє підвищити швидкість ідентифікації, а з іншого, мінімізувати використання лінії передачі даних - потенційно найбільш небезпечного для спроб незаконного доступу елементів схеми ідентифікації;

- виграє в швидкості серед інших алгоритмів при вході користувача в систему.

Недоліки

- число вхідних кодів, що ідентифікують кожного з користувачів і складають множину Ω обмежена;

- відносно великий обсяг інформації, що вимагається для опису сформованого користувачем булевого незворотного функціонального перетворення F(X);

- користувач витрачає більше часу на реєстрацію, адже йому необхідно сформувати таблиці для булевих функціональних перетворень.

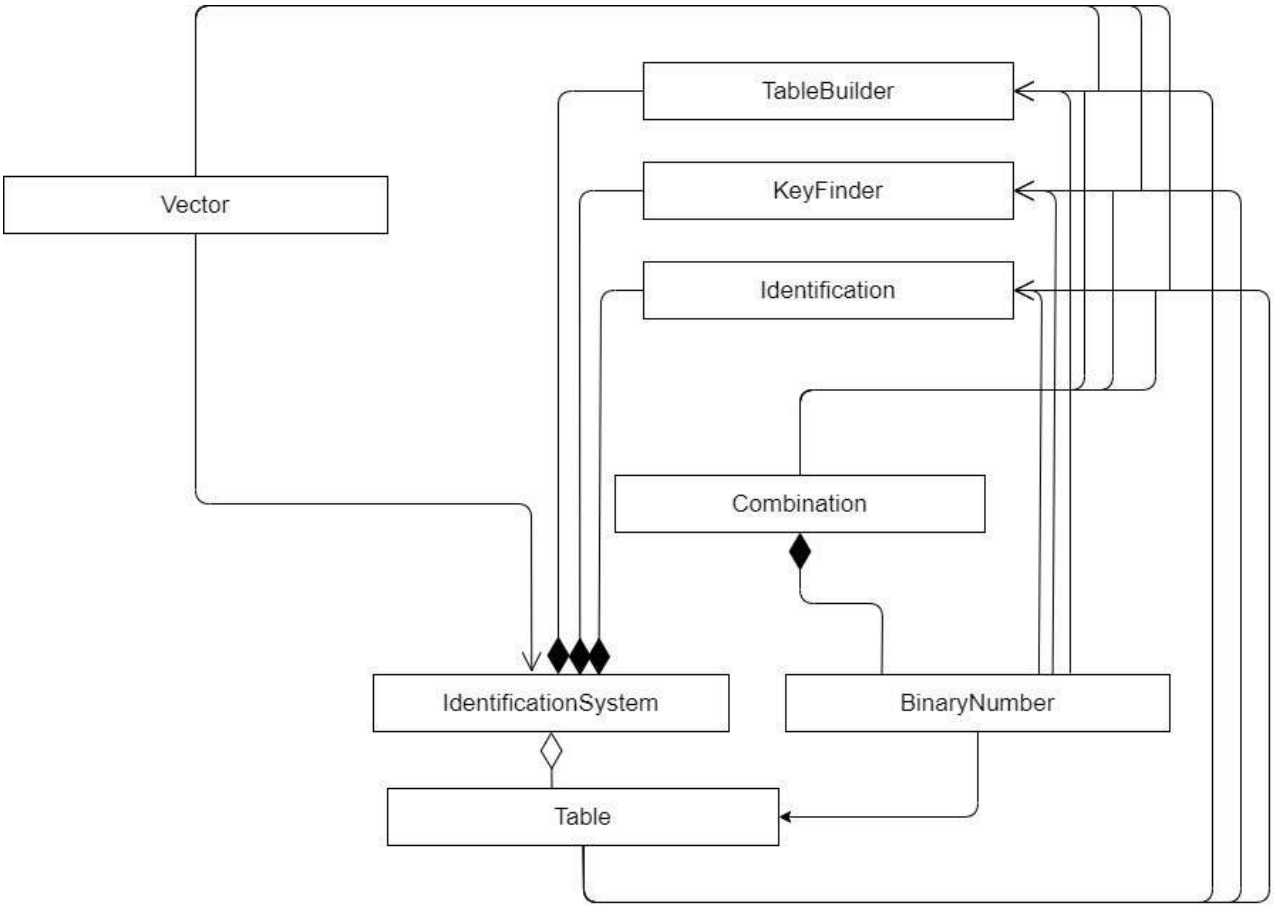
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мао В. Современная криптография теория и практика. - 2005 - 768 с.
2. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы и исходные тексты на языке С. - 2003.- № 2 – 610с.
3. C. Paar, J. Pelzl Understanding Cryptography. A Textbook for Students and Practioners. - 2010. – 382с.
4. RC-5[Електронний ресурс] – Режим доступу до ресурсу: http://crypto.pp.ua/2011/01/algoritm-rc5/
5. Алгоритмы/Хеш-функция SHA256 [Електронний ресурс] – Режим доступу до ресурсу:https://medium.com/dtechlog/%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B-%D1%85%D1%8D%D1%88-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F-sha-256-9862302f942f
6. Алгоритм RIPEMD-160 [Електронний ресурс] – Режим доступу до ресурсу: http://solutionmes.wikidot.com/crypto-ripemd
7. G. McLean Hall Pro_WPF_and_Silverlight_MVVM. - 2010. - 273с.
8. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. - 2013.- № 4 - 896с.
9. Э. Фримен, Э. Робсон Head First. Паттерны проектирования. Обновленное юбилейное издание. – 2018. - 657с.
10. System.Windows.Controls [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/dotnet/framework/wpf/controls/control-library
11. А. Жилинский Самоучитель. Microsoft SQL Server 2008. - 2015. — 213 с.
12. MS SQL Server[Електронний ресурс] – Режим доступу до ресурсу: https://www.atlantic.net/what-is-mssql/

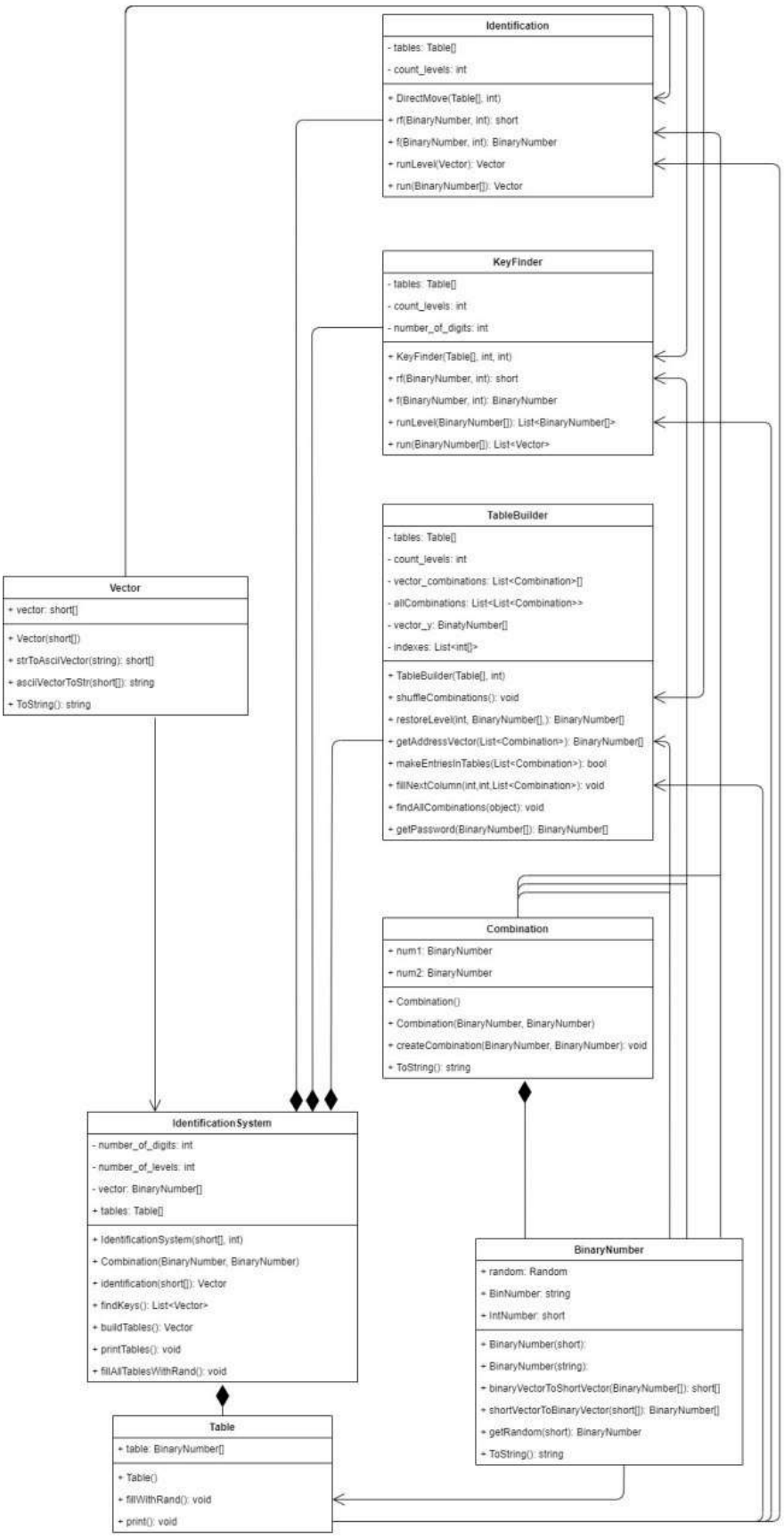
13. MS SQL Server[Електронний ресурс] – Режим доступу до ресурсу:https://metanit.com/sql/sqlserver/
14. М. Грубер Понимание SQL. - 1993. — 291 с.
15. What is Entity Framework? [Електронний ресурс] – Режим доступу до ресурсу: https://www.entityframeworktutorial.net/what-is-entityframework.aspx

Д О Д А Т К И

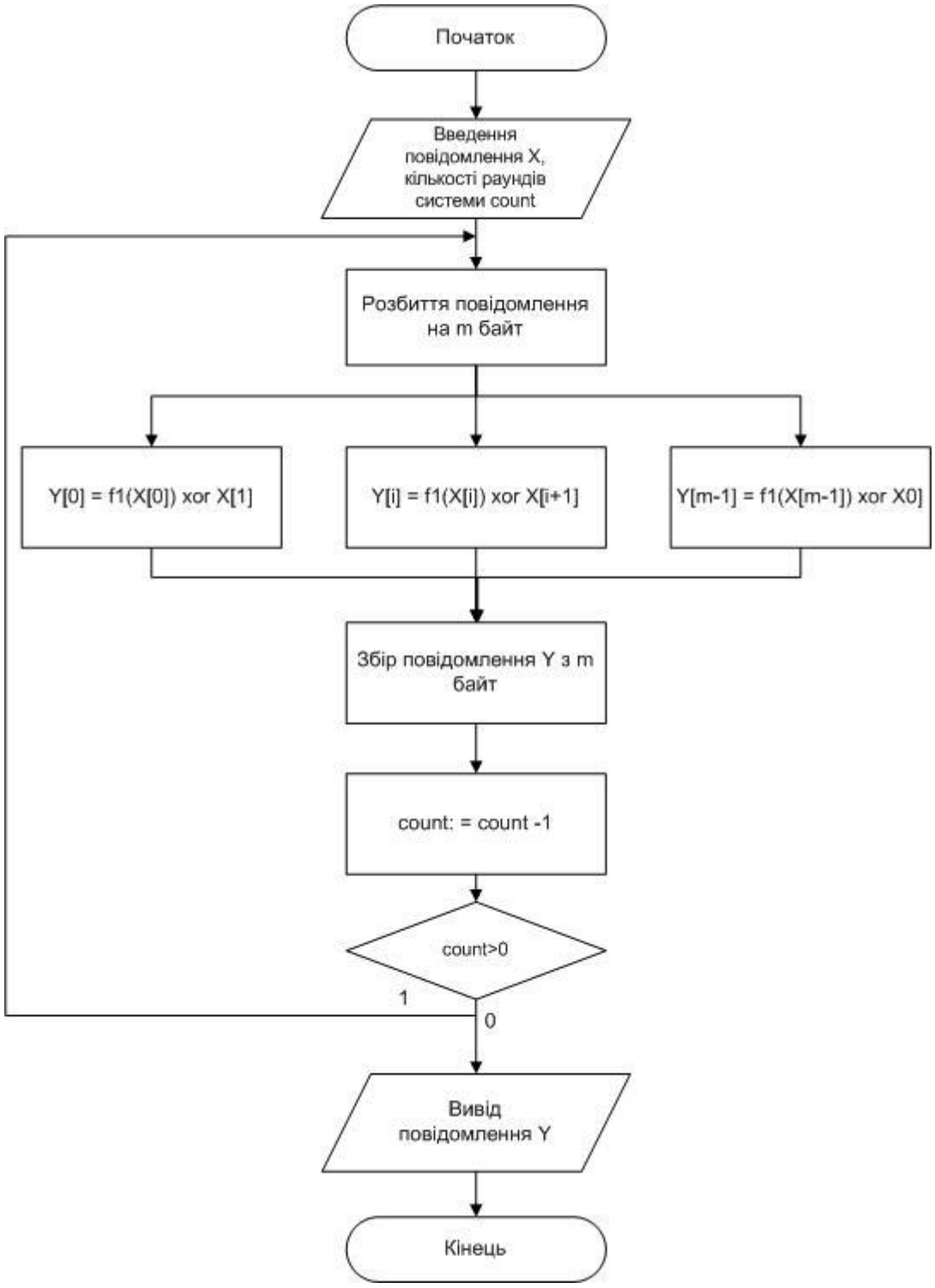
Київ – 2020 року



					ІАЛЦ.467200.004 Д1		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Климова А.С.			Структура, метод побудови незворотних булевих перетворень для криптографічно строгої ідентифікації віддалених користувачів та програмні засоби його реалізації	Літ.	Аркуш 1 / Аркушів 1
Перев.		Марковський О.П.					
Тех.контр.					Структурна схема	НТУУ "КПІ"ім. Ігоря Сікорського, ФІОТ, ІО-62	
Н.Контр.		Сімоненко В.Г.					
Затвердж.		Дичка І.А.					



					ІАЛЦ.467200.004 Д2		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Климова А.С.			Структура, метод побудови незворотних булевих перетворень для криптографічно строгої ідентифікації віддалених користувачів та програмні засоби його реалізації	Літ.	Аркуш 1 / Аркушів 1
Перев.		Марковський О.П.					
Тех.контр.						НТУУ "КПІ"ім. Ігоря Сікорського, ФІОТ, ІО-62	
Н.Контр.		Сімоненко В.Г.					
Затвердж.		Дичка І.А.			Функціональна схема		



					ІАЛЦ.467200.004 ДЗ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Климова А.С.			Структура, метод побудови незворотних булевих перетворень для криптографічно строгої ідентифікації віддалених користувачів та програмні засоби його реалізації	Літ.	Аркуш 1 / Аркушів 1
Перев.		Марковський О.П.					
Тех.контр.						НТУУ "КПІ"ім. Ігоря Сікорського, ФІОТ, ІО-62	
Н.Контр.		Сімоненко В.Г.			Принципова схема		
Затвердж.		Дичка. І.А.					

IdentificationSystem.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Client
{
    class IdentificationSystem
    {
        int number_of_digits;
        int number_of_levels;
        BinaryNumber[] vector;
        public Table[] tables;

        public IdentificationSystem(short[] vector,int count_levels)
        {
            number_of_levels = count_levels;
            this.vector = BinaryNumber.shortVectorToBinaryVector(vector);
            number_of_digits = Convert.ToString(vector.ToList().Max(), 2).Length;
            tables = new Table[vector.Length];
            for (int i = 0; i < vector.Length; i++)
                tables[i] = new Table(number_of_digits);
        }

        public Vector identification(short[] vect)
```

```
        Identification ident = new Identification (tables, number_of_levels);
        return ident.run(BinaryNumber.shortVectorToBinaryVector(vect));
    }
    public List<Vector> findKeys(short[] vect)
    {
        KeyFinder rm = new KeyFinder(tables, number_of_levels,
number_of_digits);
        return rm.run(BinaryNumber.shortVectorToBinaryVector(vect));
    }

    public Vector buildTables()
    {
        TableBuilder rm = new TableBuilder(tables, number_of_levels);
        var watch = System.Diagnostics.Stopwatch.StartNew();
        BinaryNumber[] password = rm.getPassword(vector);
        if (password == null)
            return null;
        Vector res = new
Vector(BinaryNumber.binaryVectorToShortVector(password));
        return res;
    }

    public void printTables()
    {
        for (int i = 0; i < tables.Count(); i++)
        {
            Console.WriteLine("Table" + (i+1));
            tables[i].print();
        }
```

```
        }

        public void fillAllTablesWithRand()
        {
            for (int i = 0; i < tables.Length; i++)
                tables[i].fillWithRand();
        }
    }
}
```

TableBuilder.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace Client
{
    class TableBuilder
    {
        Table[] tables;
        int count_levels;
        List<Combination>[] vector_combinations;
        List<List<Combination>> allCombinations;
        BinaryNumber[] vector_y;
        List<int[]> indexes;
```

```
        public TableBuilder(Table[] tables, int count_levels)
        {
            this.count_levels = count_levels;
            this.tables = tables;
            indexes = new List<int[]>();
        }

        public void shuffleCombinations()
        {
            for (int i = 0; i < vector_combinations.Length; i++)
            {
                for (int j = 0; j < vector_combinations[i].Count; j++)
                {
                    Combination tmp = vector_combinations[i][0];
                    vector_combinations[i].RemoveAt(0);

vector_combinations[i].Insert(BinaryNumber.random.Next(vector_combinations[i]
.Count), tmp);
                }
            }
        }
        public BinaryNumber[] restoreLevel(int level, BinaryNumber[] vector_y)
        {
            indexes.Add(new int[tables.Length-1]);
            for (int i = 0; i < indexes[level-1].Length; i++)
                indexes[level - 1][i] = 0;
            BinaryNumber[] vector_x;
            this.vector_y = vector_y;
            vector_combinations = new List<Combination>[vector_y.Length];
```

```
        Thread[] threads = new Thread[vector_y.Length];

        //find combinations for all digits
        for (int i = 0; i < vector_y.Length; i++)
        {
            threads[i] = new Thread(new
ParameterizedThreadStart(findAllCombinations));
            threads[i].Start(i);
        }

        for (int i = 0; i < threads.Length; i++)
            threads[i].Join();

        shuffleCombinations();

        //get all possible combinations
        for (; indexes[level-1][0] != vector_combinations.Count() - 1;)
        {
            allCombinations = new List<List<Combination>>();
            fillNextColumn(level, 0, new List<Combination>());

            List<Combination> correct_combination = null;
            bool answer = false;
            for (int i = 0; i < allCombinations.Count(); i++)
            {
                answer = makeEntriesInTables(allCombinations[i]);
                if (answer)
```

```
                {
                    correct_combination = allCombinations[i];
                    vector_x = getAddressVector(correct_combination);
                    if (level < count_levels)
                        vector_x = restoreLevel(level + 1, vector_x);
                }
            }
        }
        if (vector_x[0] == null)
            return vector_y;
        else
            return vector_x;
    }
    public BinaryNumber[] getAddressVector(List<Combination> combination)
    {
        BinaryNumber[] vector_x = new BinaryNumber[combination.Count()];
        for (int i = 0; i < combination.Count(); i++)
            vector_x[i] = new
BinaryNumber((short)tables[i].table.ToList().IndexOf(combination[i].num1));

        return vector_x;
    }

    public bool makeEntriesInTables(List<Combination> vect_of_combinations)
    {
        for (int i = 0; i < vect_of_combinations.Count(); i++)
        {
            int table_index = i + 1;
            if (table_index == vect_of_combinations.Count())
```

```
                table_index = 0;

            if (tables[table_index].table.FirstOrDefault(x => x != null &&
x.IntNumber == vect_of_combinations[table_index].num1.IntNumber) != null)
            {
                if
(!(tables[table_index].table[vect_of_combinations[i].num2.IntNumber] ==
vect_of_combinations[table_index].num1))
                    return false;
            }
            if (tables[table_index].table[vect_of_combinations[i].num2.IntNumber]
!= null)
                return false;
        }
        for (int i = 0; i < vect_of_combinations.Count(); i++)
        {
            int table_index = i + 1;
            if (table_index == vect_of_combinations.Count())
                table_index = 0;

            tables[table_index].table[vect_of_combinations[i].num2.IntNumber] =
vect_of_combinations[table_index].num1;
        }
        return true;
    }

    public void fillNextColumn(int level, int colIndex, List<Combination> mas)
    {
        if (colIndex == vector_combinations.Count())
```

```
        {
            allCombinations.Add(mas);
            return;
        }
        for (int i = 0; i < vector_combinations[colIndex].Count(); i++)
        {
            if (colIndex == (vector_combinations.Count() - 1) || i == indexes[level-
1][colIndex])
            {
                List<Combination> mas_i = new List<Combination>();
                mas_i.AddRange(mas);
                mas_i.Add(vector_combinations[colIndex][i]);
                fillNextColumn(level, colIndex + 1, mas_i);
            }
        }
        int j = indexes[level - 1].ToList().LastIndexOf(indexes[level -
1].ToList().LastOrDefault(x => x != (vector_combinations.Length - 1)));
        if (j != -1)
        {
            indexes[level - 1][j] += 1;
            for (int i = j + 1; i < indexes[level - 1].Length; i++)
                indexes[level - 1][i] = 0;
        }
    }

    /// <summary>
    ///  x1^x2 = result
    /// </summary>
    /// <param name="result"></param>
```

```
        /// <returns></returns>()
        public void findAllCombinations(object index)
        {
            vector_combinations[(int)index] = new List<Combination>();
            int size = tables[0].table.Length;
            for (short i = 0; i < size; i++)
            {
                Combination tmp = new Combination();
                tmp.createCombination(vector_y[(int)index], new BinaryNumber(i));
                vector_combinations[(int)index].Add(tmp);
            }
        }

        public BinaryNumber[] getPassword(BinaryNumber[] vector_y)
        {
            int level = 1;
            BinaryNumber[] vector_x = restoreLevel(level, vector_y);

            return vector_x;
        }
    }
}
```

Identification.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace Client
{
    class Identification
    {
        Table[] tables;
        int count_levels;

        public Identification (Table[] tables, int count_levels)
        {
            this.tables = tables;
            this.count_levels = count_levels;
        }

        /// <summary>
        /// find next number by address in table
        /// </summary>
        /// <param name="x"></param>
        /// <param name="index"></param>
        /// <returns></returns>
        public short rf(BinaryNumber x, int index)
        {
            BinaryNumber item = tables[index].table.ToList().FirstOrDefault(a =>
a.IntNumber == x.IntNumber);
            return (short)tables[index].table.ToList().IndexOf(item);
        }

        public BinaryNumber f(BinaryNumber x, int index)
        {
            return tables[index].table[x.IntNumber];
```

```
        }

        /// <summary>
        /// make one step by all of functions
        /// </summary>
        /// <param name="vector"></param>
        public Vector runLevel(Vector vector)
        {
            BinaryNumber[] vect =
BinaryNumber.shortVectorToBinaryVector(vector.vector);
            BinaryNumber[] result = new BinaryNumber[vect.Length];
            for (int i=0;i< vect.Length;i++)
            {
                BinaryNumber a = f(vect[i], i);
                BinaryNumber b;
                if (i == vect.Length-1)
                    b = vect[0];
                else
                    b = vect[i + 1];
                result[i] = new BinaryNumber((short)(a.IntNumber ^ b.IntNumber));
            }
            return new Vector(BinaryNumber.binaryVectorToShortVector(result));
        }

        public Vector run(BinaryNumber[] vector)
        {
            Vector x = new
Vector(BinaryNumber.binaryVectorToShortVector(vector));
            for (int i = 0; i < count_levels; i++)
```

```
            x = runLevel(x);

            return x;
        }
    }
}
```

KeyFinder.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Client
{
    class KeyFinder
    {
        Table[] tables;
        int count_levels;
        int number_of_digits;
        public KeyFinder(Table[] tables, int count_levels, int number_of_digits)
        {
            this.tables = tables;
            this.count_levels = count_levels;
            this.number_of_digits = number_of_digits;
        }

        /// <summary>
```

```
        /// find next number by address in table
        /// </summary>
        /// <param name="x"></param>
        /// <param name="index"></param>
        /// <returns></returns>
        public short rf(BinaryNumber x, int index)
        {
            BinaryNumber item = tables[index].table.ToList().FirstOrDefault(a =>
a.IntNumber == x.IntNumber);
            return (short)tables[index].table.ToList().IndexOf(item);
        }

        public BinaryNumber f(BinaryNumber x, int index)
        {
            return tables[index].table[x.IntNumber];
        }

        /// <summary>
        /// make one step by all of functions
        /// </summary>
        /// <param name="vector"></param>
        public List<BinaryNumber[]> runLevel(BinaryNumber[] vector)
        {
            List<BinaryNumber[]> parent_keys = new List<BinaryNumber[]>();

            //create special tables
            Table[] t = new Table[vector.Length];
            for (int i = 0; i < t.Length; i++)
                t[i] = new Table(number_of_digits);
```

```
            for (int i = 0; i < t.Length; i++)
            {
                for (int j = 0; j < t[i].table.Length; j++)
                {
                    t[i].table[j] = new BinaryNumber((short)(j ^ vector[i].IntNumber));
                }
            }

            //find parent keys
            for (short i = 0; i < t[0].table.Length; i++)
            {
                BinaryNumber[] key = new BinaryNumber[vector.Length];
                BinaryNumber index = new BinaryNumber(i);
                key[0] = new BinaryNumber(rf(index, 0));
                for (int j = 1; j < vector.Length; j++)
                {
                    key[j] = t[j - 1].table[index.IntNumber];
                    index = f(key[j], j);
                }

                if (key[0].IntNumber == t[vector.Length -
1].table[index.IntNumber].IntNumber)
                    parent_keys.Add(key);
            }

            return parent_keys;
        }
```

```
        public List<Vector> run(BinaryNumber[] vector)
        {
            List<BinaryNumber[]> keys = new List<BinaryNumber[]>();
            keys.Add(vector);
            for (int i = 0; i < count_levels; i++)
            {
                List<BinaryNumber[]> parentkeys = new List<BinaryNumber[]>();
                foreach (var item in keys)
                {
                    parentkeys.AddRange(runLevel(item));
                }

                keys = parentkeys;
            }

            List<Vector> vector_keys = new List<Vector>();
            foreach (var item in keys)
            {
                vector_keys.Add(new
Vector(BinaryNumber.binaryVectorToShortVector(item)));
            }

            return vector_keys;
        }
    }
}
```

Combination.cs

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Client
{
    class Combination
    {
        public BinaryNumber num1 { get; set; }
        public BinaryNumber num2 { get; set; }

        public Combination()
        {}

        public Combination(BinaryNumber num1, BinaryNumber num2)
        {
            this.num1 = num1;
            this.num2 = num2;
        }

        public void createCombination(BinaryNumber result, BinaryNumber num)
        {
            num1 = num;
            num2 = new BinaryNumber((short)(result.IntNumber ^ num.IntNumber));
        }

        public override string ToString()
        {
```

```
            return $"{num1.BinNumber} - {num2.BinNumber}";
        }
    }
}
```

Table.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Client
{
    class Table
    {
        public BinaryNumber[] table { get; set; }

        public Table(int size)
        {
            table = new BinaryNumber[(int)Math.Pow(2,size)];
        }

        public void fillWithRand()
        {
            for (int i = 0; i < table.Length; i++)
            {
                if (table[i] == null)
                {
```

```
                BinaryNumber num =
BinaryNumber.getRandom((short)table.Length);
                while (table.FirstOrDefault(x=> x!= null && x.IntNumber ==
num.IntNumber) != null)
                    num = BinaryNumber.getRandom((short)table.Length);
                table[i] = num;
            }
        }
    }
    public void print()
    {
        foreach(var item in table)
            Console.WriteLine(item);
    }
  }
}
```

BinaryNumber.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Client
{
  class BinaryNumber
  {
    short number;
    public static Random random = new Random();
```

```
        public short IntNumber { get { return number; } set { number = value; } }
        public string BinNumber { get { return Convert.ToString(number, 2); } set
{number = Convert.ToSByte(value, 2); } }

        public BinaryNumber(short num)
        {
            IntNumber = num;
        }
        public BinaryNumber(string num)
        {
            BinNumber = num;
        }

        /// <summary>
        /// generate one random binary number
        /// </summary>
        /// <param name="max"></param>
        /// <returns></returns>
        public static BinaryNumber getRandom(short max)
        {
            short result = (short)random.Next(max);
            return new BinaryNumber(result);
        }

        /// <summary>
        /// convert vector of int to vector of binary numbers
        /// </summary>
        /// <param name="vector"></param>
```

```
        /// <returns></returns>
        public static BinaryNumber[] shortVectorToBinaryVector(short[] vector)
        {
            BinaryNumber[] binary_vector = new BinaryNumber[vector.Length];
            for(int i=0;i<binary_vector.Length;i++)
                binary_vector[i] = new BinaryNumber(vector[i]);

            return binary_vector;
        }

        /// <summary>
        /// convert vector of binary numbers to vector of int
        /// </summary>
        /// <param name="binary_vector"></param>
        /// <returns></returns>
        public static short[] binaryVectorToShortVector(BinaryNumber[]
binary_vector)
        {
            short[] vector = new short[binary_vector.Length];
            for (int i = 0; i < vector.Length; i++)
                vector[i] = binary_vector[i].IntNumber;

            return vector;
        }

        public override string ToString()
        {
            return $"{IntNumber}";
        }
```

```
    }
}
```

Vector.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Client
{
    class Vector
    {
        public short[] vector { get; set; }

        public Vector(short[] vect)
        {
            vector = vect;
        }

        public static short[] strToAsciiVector(string pass)
        {
            short[] vec = new short[pass.Length];

            for (int i=0;i<pass.Length;i++)
            {
                vec[i] = (short)pass[i];
            }
```

```
            return vec;
        }

        public static string asciiVectorToStr(short[] vec)
        {
            byte[] vect = new byte[vec.Length];
            string pass = "";
            for (int i = 0; i < vec.Length; i++)
            {
                pass += Convert.ToChar(vec[i]);
            }
            return pass;
        }
        public override string ToString()
        {
            //return asciiVectorToStr(vector);
            return String.Join(" ", vector);
        }
    }
}
```